Vik's Software Guide

Release 2019

Viktor Kis

Sep 20, 2020

CONTENTS

Acknowledgements: Content made better by K. Kim!

CHAPTER

ONE

ABSTRACT

This guide is a collection of references and examples for common commands/examples mainly structured around python, shell (unix/linux/cmd), Microsoft Office related vba and other similar frameworks. All documentation are open source, and can be found on public domain. It was created to help others, in hope that at least one user of these docs may also contribute to a better tomorrow the best way they can.

The objective is to layout concepts and examples step by step in a clear and concise manner with emphasis on the "gotchas".

Note: Concepts laid out here are not meant to be read from top to bottom. These are merely buckets of concepts. For beginners, start with the 10min intro topics. Future revisions may include a beginner/intermediate/advanced walkthrough.

CHAPTER

TWO

SUPPORT CONTENT CREATOR

If you enjoyed this library, please consider supporting its creators! Help Today

Feel free to drop a comment on the github page: https://github.com/PydPiper/viks_SoftwareGuide

2.1 Shell Guide

2.1.1 Linux Quick Start Guide

Setup

- To add python to PATH: `export PATH="/C/Users\vkisf\AppData\Local\Programs\Python\Python38:$PA`
- To add python scripts (pytest etc.) to PATH: `export PATH="/C/Users\vkisf\AppData\Local\Programs\Python\Python38\Scripts:$PATH"`
- To add java to PATH: `export PATH="/C/Users/vkisf/AppData/Local/Programs/AdoptOpenJDK/jdk-11.0.8.10-hotspot/bin:$PATH"`
- To set an alias: ‘‘alias python38=”winpty /C/UsersvkisfAppDataLocalProgramsPythonPython38python.exe”
- To ssh: `alias ssh_server1="ssh username@servername.com"`

General

- To clear terminal window text: `clear`
- To copy text from terminal: simply highlight the text
- To paste text into terminal: `Shift` + `Insert`
- To change directories: `cd folder1/folder2`
- To back out a directory: `cd ..` then back out 2 and so on: `cd ../..`
- To back out to home directory: `cd ~`
- To create a directory: `mkdir folder1`
- To create a file: `touch file.txt`
- To concatenate 2 files: `cat file1.txt file2.txt > file2.txt`
  - Pipe to a file: `>`
  - To append to a file: `>>`

- To to print file content to terminal: `cat file1.txt`
- To execute a file: `./file1.txt`
- To remove a file: `rm file1.txt`
- To remove all files under a folder: `rm -rf folder1`
- To remove a folder: `rmdir folder1`
- To rename a file/folder: `mv file1.txt file2.txt`
- To get current working directory: `pwd`
- To get list of files/folders in your current working directory: `dir -la` or `ls -la` (-l for long desc, -a for hidden)
- To search for text in files/folders: `grep -r "text" *` (the "*" is a wild card)

File Permissions

- To get list of files/folders with permission levels in your current working directory: `dir -la`
- To change file/folder permissions: `chmod -R u=rwx folder`
  - `-R` is recursively change all files/folder under the given folder
  - `u` is for "user", `g` for "group", `o` for "other" and `a` for "all"
  - `r` for "read", `w` for "write", `x` for "execute"
- To change group of a file/folder: `chgrp new_groupname file.txt`
- To check which groups you belong to: `groups`

Scripting

- To print datetime: `date +%m%d%y` (note lower case is short form, upper case is long form)
- To declare a variable in a script: `variable1 = "this"` and to call it `$variable1`

2.2 Python Guide

2.2.1 10min - Python Starter Kit

This starter kit was meant for readers that have never used python and have very little knowledge of programming concepts.

Installation - Windows Guide Only

1) Download the latest version of python from python.org
2) During installation all the default selection will work, but pay attention to where python is being installed. The newer versions on windows will be placed under: `c/users/yourusername/AppData/Local/Programs`
3) Once python is installed browse to folder it was installed in, something like `c/users/yourusername/AppData/Local/Programs/Python38-32` and inside you will see a `python.exe`

4) Add the python folder to your windows `PATH` so that you can pull up python from any terminal (cmd, powershell, bash).

    4.1) Hit the windows start menu (bottom left windows icon)

    4.2) Type `environment` and you are looking for `edit environment variables for your account`

    4.3) Highlight `Path` and hit `edit`

    4.4) A new window will open, hit `new` and paste in your path where python was installed `c/users/yourusername/AppData/Local/Programs/Python38-32`

    4.5) Hit `OK` on both windows and you are good to go!

5) Running python: pull up a terminal (start menu > cmd > enter) and type `python`. If the terminal hangs on windows try `python -i` where -i is interactive, or `winpty python`

Things you should know about:

- Python is manually downloaded from python.org. There is no update button to get a newer version, you will have to go back to python.org and download a new version manually again.
- When installing Python, you are also installing a python package installer (pip) that unlocks python's superpowers. Packages can be imported with a single line of a code and before you know it you are scraping the web, working on excel/text files or performing machine learning with only 10 lines of code.
- Everything in python is about versions. Python has a version, under it your pip has a version, under that your packages will have versions.

Run your first Script

You have python installed and it works. Now you can type away at a python session in a terminal but once you close the terminal, your code will also be gone. So instead we can write a script can you can call any number of times:

1) create a script file (lets say on your desktop): Right Mouse Button > New > Text Document
2) rename it `myscript.py`
3) open it with your text editor (notepad++ is a decent pick)
4) type out your python code and save, example:

```
print("Hello World!")
```

5) run your script by typing `python myscript.py` in your terminal(your terminal has to be in the same folder). Start Menu > type "cmd" > enter > then in the terminal `cd Desktop` then try `python myscript.py`

Python Highlevel Concepts

Note: ">>>" is not part of any code, it is simply shown here to distinguish between code written and its output results. (ie. do not copy/write lines containing ">>>")

Note: the following names are reserved for python internals, and should not be used as variable names `false none true and as assert break class continue def del elif else except finally for from global if import in is lambda nonlocal not or pass raise return try while with yield`

Basics

- code comment: `# this is a comment`
- define a variable (no declaration needed!): `a = 5`
- understand your basic types:
  - int:`1` or `2342134`
  - float: `1.0` or `2342134.12341234`
  - string: `"this"` or `'that'` both valid but double quotes is better for `"it's a nice day"`
  - list (arrays if you prefer): 1D `[1,2,3]` 2D `[[1,2,3],[10,20,30],[100,200,300]]`
  - there are a lot more but these are the basics

What can I do with integers/floats (math)

```
a = 5
b = 10
c = a + b
print(c)
>>> 15
# add, subtract, multiply, divide, power
5 + 10 - 10 * 5 /5 ** 2
>>> 13.0
```

What can I do with strings

- split up text

```
a = 'this is a string'
b = a.split(" ") # split text base on " " single spaces
b
>>> ['this', 'is', 'a', 'string']
```

- replace characters

```
a = 'this is a string'
b = a.replace('s','S')
b
>>> 'thiS iS a String'
```

- add two strings

```
a = 'this'
b = 'that'
c = a + b
c
>>> "thisthat'
# or use join, note items have to be in square brackets
d = ' '.join([a,b]) # join "a" and "b" with a " " space
>>> 'this that'
```

- sub-strings (slicing)

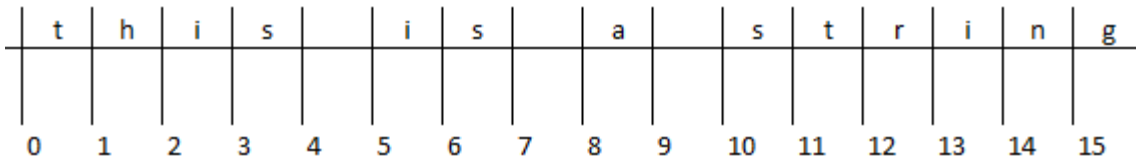


Fig. 1: List slicing

```
a = 'this is a string'
a[0] # index to a character (python indexing start at 0)
>>> 't'
b = a[0:4] # give me the characters from index 0 to start-of index 4, t=0,h=1,i=2=s=3,
↪4=' '
b
>>> 'this'
```

What can I do with lists

- indexing

```
a = [10,20,30]
a[0] # python indexing starts at 0
>>> 10
a[0:2] # from index 0=10, to right before index 2=30 so that's 20
>>> [10,20]
```

- add to a list

```
a = [] # empty list
a.append(10) # append one at a time
a += [20,30] # add another list to it
a
>>> [10,20,30]
```

- 2D array (really just a nested list)

```
x = [10,20,30] # 3 x-coordinates
y = [40,50,60] # 3 y-coordinates
myarray = list(zip(x,y))
myarray
>>> [(10, 40), (20, 50), (30, 60)]
myarray[1] # what is the x,y -coordinate of point 2 (note again python index starts␣
↪from 0)
>>> (20,50)
myarray[1][0] # what is the x-coordinate of point 2
>>> 20
myarray[1][1] # what is the y-coordinate of point 2
>>> 50
```

How to write logic loops (if, for, while)

equal: ==, not equal: `!=`, and: `and`, or: `or`

- if statements

```
if 1 == 1 and 1 == 2:
    print('1 is equal to 1 and also equal to 2')
elif 1 != 1:
    print('1 is not equal to 1')
else:
    print('none of the conditions were true')
```

- for loop

```
mylist = [10,20,30]
for item in mylist:
    print(item)
>>> 10
>>> 20
>>> 30
```

- while loop

```
i = 0
while i < 3:
    print(i)
    i += 1
>>> 0
>>> 1
>>> 2
```

How to write functions

```
# define function with 2 inputs
def myfunc(input1, input2):
    result = intput1 + input2 + 10
    return result

# call a function with inputs 1,2
func(1,2)
>>> 13
```

How do I read/write files

- read a file

```
# container for lines of text out of our file
lines = []

# use the python builtin function "open" to start streaming a file for read "r"
with open('test.txt', 'r') as f:
    while True:
        # read each line in a file
        line = f.readline()
        # add each line to our container
        lines.append(line)
        # at the end of the file, line=""
```

(continues on next page)

(continued from previous page)

```
        # in which case we stop reading the file and break out of the loop
        if not line:
            break
```

- write a file

```
# writing is very similar, except we "w" for write
with open('test2.txt', 'w') as f:
    f.write('Hello World')
```

2.2.2 Links - Python Awesome List

This is a collection of great resources ranging from educational links, articles, tutorials, blogs/podcasts, and current events centric around python. None of links here are paid advertise, just simply a great collection of resources.

Educational Online

- Python Docs: Always start with official docs and then branch out if there is not enough information there to figure a problem out
- w3school: Great collection of high level topics with examples
- realpython: By far one of the greatest collection of example driven tutorials with a great community
- tutorialspoint: Similar to w3school a collection of high level topics
- Learn Python The Hard Way: (paid) Great set of resources with lots of examples
- Geeks for Geeks: High level python topics

Educational Books

- thinkpython: (free) Great book also available in paper form
- Dive Into Python3: (free) Python3 centric with notes about how it is different to python 2
- Functional Python Programming: (free) Python topics explained from a functional programming stand point

Current Events

- Talk Python Podcast Michael Kennedy's python podcast. Michael interviews folks from all background; software developers, mathematicians, scientists, business, and more on how they have enriched their lives with python.
- Pycascades Conference Pacific north west conference all about python.

2.2.3 builtin - Class (Object Oriented Programming)

Class object versus Class object instance

- define a class

```
class Circle():
    # to set initialization parameters
    #  (these are unique attributes to each instance of the Circle class)
    def __init__(self, radius):
        # self.radius is called a INSTANCE ATTRIBUTE
        self.radius = radius
        # WHAT IS "SELF": think of self as the instance of the class; ex:
        #   at runtime: circle1 = Circle(5), the class instance "circle1" is self
        #   this is why we are able to call circle1.radius which is
        #   analogous to self.radius of that class instance
```

- # common mistake #1: AttributeError: type object 'Circle' has no attribute 'radius'

```
Circle.radius
>>> AttributeError: type object 'Circle' has no attribute 'radius'
# this occurs because we did not initialize an instance of Circle

# fix:
Circle(radius=10).radius
>>> 10
```

- we can also save an INSTANCE of Circle as a variable (how its commonly used)

```
circle1 = Circle(10)
circle2 = Circle(20)
circle1.radius
>>> 10
circle2.radius
>>> 20
```

Class Method (same as a function)

- define a class with a METHOD

```
class Circle():
    def __init__(self, radius):
        self.radius = radius

    # to define a METHOD (unique name to classes, same concept as a function)
    def area(self):
        return 3.14 * self.radius ** 2
```

- common mistake #2: TypeError: area() missing 1 required positional argument: 'self'

```
Circle.area()
>>> TypeError: area() missing 1 required positional argument: 'self'
# same as above, this occurs because we did not initialize an instance of Circle

# fix:
Circle(radius=10).area()
>>> 314.0
```

- assigned to a variable

```
circle1 = Circle(10)
circle1.area()
>>> 314.0
```

Class Attribute vs. Instance Attribute

- define a class with CLASS and INSTANCE ATTRIBUTES

```
# classes are great with their simple dot completion attributes, however
#  results can be very different than expected when using different attribute types

class Circle():
    # CLASS ATTRIBUTE: same for all instances of the class
    PI = 3.14           # immutable CLASS ATTRIBUTE
    classlist = [1,]    # mutable CLASS ATTRIBUTE

    def __init__(self, radius):
        # INSTANCE ATTRIBUTE: unique to each instance of the class
        self.radius = radius
```

- define 2 instances of the parent class Circle

```
# lets create 2 instances of the class Circle
circle1 = Circle(radius=5)
circle2 = Circle(radius=10)

# note that circle1 and 2 both have a CLASS ATTRIBUTE .PI that is the same
circle1.PI
>>> 3.14
circle2.PI
>>> 3.14
# but their INSTANCE ATTRIBUTE is unique to each instance of the class Circle
circle1.radius
>>> 5
circle2.radius
>>> 10
```

- Updating CLASS ATTRIBUTES

```
# CLASS ATTRIBUTES are connected to all instances of that class,
#   we can change all of them at once by modifying the master CLASS ATTRIBUTE
circle1.PI
>>> 3.14
circle2.PI
>>> 3.14
# now lets update both from the parent class Circle
Circle.PI = 50
circle1.PI
>>> 50
circle2.PI
>>> 50
```

- Updating CLASS ATTRIBUTES the wrong way!

```
# IMPORTANT: python lets you do whatever you like, but with such power comes␣
↪consequences
#   ex: the ability to overwrite a CLASS ATTRIBUTE of a class instance like circle1
#   note that prior to modifying .PI CLASS ATTRIBUTE has the same ID for all instances
id(circle1.PI)
>>> 72539584
id(circle2.PI)
>>> 72539584
# now when we overwrite .PI we are actually changing the .PI attribute from CLASS to␣
↪INSTANCE ATTRIBUTE
circle1.PI = 3
id(circle1.PI)
>>> 1865210064
# also note that now instances DO NOT share the same .PI CLASS ATTRIBUTE any more
circle2.PI
>>> 3.14


# now lets see what happens with a mutable CLASS ATTRIBUTE
id(circle1.classlist)
>>> 71716696
id(circle2.classlist)
>>> 71716696
# similar to PI, classlist shares the same ID between classes, but now updating one
#   also updates all because the ID stays the same for mutable objects
circle1.classlist += [2]
circle1.classlist
>>> [1,2]
circle2.classlist
>>> [1,2]       # circle2 instance was also updated!
```

Class Methods (method, staticmethod, classmethod)

- define a class with a METHOD, STATICMETHOD, and CLASSMETHOD

```
# class methods are analogous to function definitions, except they are tied to a class
class Circle():
    def __init__(self, radius):
        self.radius = radius

    # This is a simple METHOD: methods take at least 1 argument "self" and does␣
↪something with it
    def area(self):
        return 3.14 * self.radius ** 2

    # This is a STATICMETHOD: a static method does not depend on "self"
    #   or more explicitly stating, any unique definition of the class instance
    @staticmethod
    def color(color='black'):
        return 'the color of the circle is: ' + color

    # This is a CLASSMETHOD: a class method takes at least 1 argument "cls" and
    #   it usually returns a new altered instance of the class
    # What is really special about a class method is that the
    #   user is able to call it without instancing the class (see example below)
    @classmethod
```

(continues on next page)

(continued from previous page)

```
    def from_dia(cls, diameter):
        # cls under the hood calls Circle.__new__() that creates a new instance of␣
↪the class Circle
        # with new __init__ definition that is: diameter/2
        return cls(diameter / 2)
```

- Call/use a METHOD

```
circle1 = Circle(radius=5)
# call a regular METHOD via
circle1.area()
>>> 78.5
```

- Call/use a STATICMETHOD

```
circle1 = Circle(radius=5)
# call a STATICMETHOD
circle1.color()
>>> 'the color of the circle is: black'
```

- Call/use a CLASSMETHOD. Define a Circle by diameter (note that the class is never instanced, ie: "Circle()") circle2 is now instanced via CLASSMETHOD, and all of the regular functionality is available

```
circle2 = Circle.from_dia(diameter=10)
circle2.radius
>>> 5.0
circle2.area()
>>> 78.5
```

Double underscore methods (dunder)

- define a class with `__init__`, `__repr__`, `__call__`

```
class Circle():
    # INIT: initialize a class instance with parameters
    def __init__(self, radius):
        self.radius = radius

    # REPR: string representation of a class (instead of the default "Circle object␣
↪at 0x23423423"
    def __repr__(self):
        return "Circle Class"

    # CALL: returns call to the class instance
    def __call__(self, *args, **kwargs):
        print(args)
        args = args if args else ("",)
        print(args)
        return "this is a call on the class, " + len(args)*"{},".format(*args)

    # ADD: defines what to do with a "+" operator
    #  note: operators always work from left, ie: Circle + 10
    #        the "+" operator is actually calling __add__ on Circle
    def __add__(self, arg):
        print("you tried to add to class Circle")
```

(continues on next page)

(continued from previous page)

```
        return arg + self.radius

    def __subtract__(self, arg):
        return "you tried to subtract from class Circle"

    def __mul__(self, arg):
        return "you tried to multiply class Circle"

    def __truediv__(self, arg):
        return "you tried to divide class Circle"

    # you can have the operator read from the right as well, this is useful if you
    #  tried to add: 10 + Circle, by default python will try to read from left but
    #  has no idea how to add a "int" + "class" so then it will look to the right and
    #  see if it has a "radd" definition, the "r" can be defined for all other math␣
↪operators
    def __radd__(self, arg):
        print("addition with right operator")
        return arg + self.radius

    # to evaluate Circle[arg] sequence
    def __getitem__(self, arg):
        return [self.radius]
```

- call/use `__init__` (class instance initialization)

```
# INIT call/use
circle1 = Circle(radius=5)
```

- call/use `__repr__` (class text representation)

```
circle1 = Circle(radius=5)
# REPR call/use
circle1
>>> "Circle Class"
# REPR call/use
str(circle1)
>>> "Circle Class"
```

- call/use `__call__` (call return of the class)

```
circle1 = Circle(radius=5)
# CALL call/use
circle1()
>>> "this is a call on the class, ,"
circle1(1,2)
>>> "this is a call on the class, 1,2"
```

- call/use `__add__` and other math dunder's

```
circle1 = Circle(radius=5)
# ADD call/use
circle1 + 5 # here __add__ gets called
>>> "you tried to add to class Circle"
>>> 10 # radius + 5
```

(continues on next page)

(continued from previous page)

```
# now a right operation, since int doesnt know how to add Circle, but Circle does
5 + circle1 # int + Circle returns an error, then python tried from right: __radd__
↪gets called
>>> "addition with right operator"
>>> 10
```

Subclassing - to extend functionality of a class

Take Circle class for instance, it has a method to calculate area now lets say Circle is locked down as a class by another coder and we cannot change it we dont want to start from scratch and rebuild Circle, but we do want to add functionality we can do this with subclassing

- define a parent class and a subclass (a subclass inherits functionality of a parent class)

```
# here is the original Circle Class
class Circle():
    def __init__(self, radius):
        self.radius = radius

    def area(self):
        return 3.14 * self.radius ** 2


# now lets create a custom Class that inherits functionality from Circle
class CustomCircle(Circle):
    def halfarea(self):
        # note that we depend on Circle having a method called area()
        #   but the method itself is not defined here in CustomCircle
        #   it is INHERITED
        return self.area() / 2
```

- using a subclass

```
# lets create an instance of our custom class
circle1 = CustomCircle(radius=10)
# note that we still have access to methods from Circle (it is INHERITED)
circle1.area()
>>> 314.0
# but we also have a new custom functions from CustomCircle
circle1.halfarea()
>>> 157.0
```

Trick - Print the docstring of a class/method

```
class Circle():
    """
    Class docs
    """

    def __init__(self):
        """
        Instance docs
        """
```

(continues on next page)

(continued from previous page)

```
        pass

    def func(self):
        """
        Method docs
        """
        pass

Circle.__doc__
>>> "Class docs"
Circle.__init__.__doc__
>>> "Instance docs"
Circle.func.__doc__
>>> "Method docs"
```

Trick - Testing that a class has a method (compile time)

```
assert hasattr(Circle, "area"), "The class Circle doesnt have required method area"
```

Trick - Access a class's attribute by its string name

```
class A():
    self.attr1 = []

getattr(A,'attr1')
```

Trick - How to create multiple levels of attributes

There are a lot of lessons on object oriented programming that talk about the big concepts, then you go to create something in practice and realize that you are still missing some fundamentals. For me, this was the case on nested attributes for a long time. How do I create a class with multiple levels of attributes?

- Lets say we want have a database class (storage class), and each time is index by an `ID` but under each `ID` we would like to call more attributes, ie: `db.ID[1].att1`. Let's see how that can be done:

```
# database class
class Database():
    def __init__(self):
        # initialize the ID container as a dictionary
        self.ID = {}

    # create a method of adding new items to the database
    def additem(self,ID,att1,att2):
        # update database dictionary by calling another class "Attributes"
        #  this says: Database.ID[#] returns the class Attributes,
        #  that can then be called for it's attributes ".att1", ".att2"
        self.ID.update({ID: Attributes(att1,att2)})


# 2nd level nested attributes class
class Attributes():
```

(continues on next page)

(continued from previous page)

```
    def __init__(self,att1,att2):
        self.att1 = att1
        self.att2 = att2


# lets see how it works in pratice

# define a instance of the database
db = Database()
# add a few items
db.additem(1,'att1 from ID1', 'att2 from ID1')
db.additem(2,'att1 from ID2', 'att2 from ID2')
# now to call it nested
db.ID[1].att1
>>> 'att1 from ID1'
db.ID[2].att2
>>> 'att2 from ID2'
```

Trick - Create multiple instances of a class based on initial input

This is really useful when a class __init__ is setup to take a single value input (like an ID, but instead a range of IDs were given) and we would like to create multiple unique classes out of each ID separately.

```
# take a class for instance that is a storage of attributes
# its unique identifier is set by an attribute ID, but
# a user would like to define multiple classes at the same time - what do we do

class Signal():

    # note __init__ is called after __new__ via super
    def __init__(self, ID, A, B, C):
        print("initialized Signal")
        self.ID = ID
        self.A = A
        self.B = B
        self.C = C

    # called before __init__
    def __new__(cls, ID, *args, **kwargs):
        # check if ID entered was a range, if so, split them apart
        if type(ID) is list:
            print("muti-ID identified")
            return cls.split_IDs(ID, *args, **kwargs)
        else:
            # this says: from the class Signal create an instance (ie: call __init__)
            print("creating instance ID = ", ID)
            # note that .__new__(cls) only has cls as input, ID, A, B, C are not␣
↪entered
            # (but they are buffered over to the __init__ automatically
            return super(Signal, cls).__new__(cls)

    @classmethod
    def split_IDs(cls, ID, *args, **kwargs):
        # return a list of Singal instances all with the same attributes A,B,C but␣
↪unique single IDs
```

(continues on next page)

(continued from previous page)

```
        print("creating a list of unique Signal instances")
        # note that each cls call here for each uniqueID in ID calls __new__ with␣
↪ID=uniqueID as input
        # therefore this call goes to the "creating instance" logic
        return [cls(uniqueID, *args, **kwargs) for uniqueID in ID]

# now let's test it for a single ID input:
single_signal = Signal(ID=1,A=10,B=20,C=30)
>>> "creating instance ID = 1"
>>> "initialized Signal"

# now for multi-ID input
list_signal = Signal(ID=[1,2],A=10,B=20,C=30)
>>> "muti-ID identified"
>>> "creating a list of unique Signal instances"
>>> "creating instance ID = 1"
>>> "initialized Signal"
>>> "creating instance ID = 2"
>>> "initialized Signal"
```

2.2.4 builtin - Tuples, Lists, Sets, Dictionaries

There are 4 common collectors available to the users in python; Tuples, Lists, Sets, and Dictionaries. Each of them have unique features that work best for different objectives.

Tuples

Tuples are immutable (once a value is added, its ID cannot be changed), see *builtin - Mutable/Immutable*. These are great for ensuring that values stay consistent throughout your code. Note that there is a subtlety here though; a tuple `test=(1,[2,3])` cannot change its IDs, however there is a list within the tuple that can change its internal values (therefore the tuple is not, truely immutable when it contains mutable objects).

- Syntax

```
a = (1,2,3)
# index an item
a[0]
>>> 1 # note first item of any container in python starts at 0
a[-1] # negative indexing is from the end
>>> 3
```

- Add to a Tuple

```
a += (4,)
a
>>> (1,2,3,4)
```

- Get a subset of a tuple

```
a[2:]
>>> (3,4)
```

- Method of Tuple: "count" occurrence of an item
- Method of Tuple: "index" finds the index for a given value

```
# there are 2 methods available on a tuple
# count: count occurrence of a item within a tuple
a.count(4)
>>> 1
# index: find the index for a given value
a.index(4)
>>> 3
```

Lists

Lists are extremely useful because you can change them on the fly (mutable object, see more on *builtin - Mutable/Immutable*). However, there is a fine line between when a List should be used over Dictionaries. You can just about do everything with a Dictionary that you can do with a List, and when Lists start to look double, triple+ nested - a Dictionary should be looked at for code readability/use.

See *List Comprehensions (alt for loops)* for list comprehensions, a 1-liner code that replaces a simple for loop with an optional if statement.

- Syntax

```
a = [1,2,3]
# index on item
a[0] # note first item of any container in python starts at 0
>>> 1
a[-1] # negative indexing is from the end
>>> 3
```

- add/remove to a list

```
a = [1,2,3]
# add to list
a += [4,]
# or
a.append(5)
a
>>> [1,2,3,4,5]
# remove an item from the list
a.remove(3)
a
>>> [1,2,4,5]
# remove item by index value
a.pop(3)
>>> 5
a
>>> [1,2,4]
```

- list slicing (sublist from list, reverse order, skip content)

```
a = [1,2,3,4,5]


# get a subset of a list
a[2:]   # this reads, [from starting index item included, to end index item NOT␣
↪included]
# by not specifying the end index, we get from index 2 all the way to the end of the␣
↪list
```

(continues on next page)

(continued from previous page)

```
>>> [3,4,5]

# negative indexing
a[:-2]  # give me everything from start to 2 indexes before the end
>>> [1,2,3]

# reverse order
a[::-1]
>>> [5,4,3,2,1]

# skip every 2 for example
a[::2]
>>> [1,3,5]
```

- define multiple variables on 1 line

```
# define multiple variables on same line
mylist = [1,2,3]
a, b, c = mylist
a
>>> 1
b
>>> 2

# also good for initializing variables
a, b, c = [""]*3 # will all be empty strings
```

List - Copy

See also *builtin - Copy True/Shallow/Deep* for more information.

- true copy -> same ID, changing the index of one, changes the other

```
a = [1,2,3]
b = a
id(a) == id(b)
>>> True
b.append(100)
b
>>> [1,2,3,100]
a
>>> [1,2,3,100]
```

- shallow copy -> new list ID, however the values are the same object ID

```
nested = [1,2]
a = [nested,3,4]
b = a[:] # this
id(b) == id(a)
>>> False
# however note that altering a MUTABLE value changes the value on both "a" and "b"
nested.append(100) # note that append is alters the list, but does not change its id
b
>>> [[1,2,100],3,4]
a # now note, that "a" also changed - this is called a shallow copy
>>> [[1,2,100],3,4]
```

- deep copy -> new list ID, and new content IDs

```
import copy as cp
nested = [1,2]
a = [nested,3,4]
b = cp.deepcopy(a) # note that this is a slow process, for optimization look for␣
↪deepcopy first
nested.append(100)
a
>>> [[1,2,100],3,4]
b
>>> [[1,2],3,4] # nested is no longer linked in a deepcopy to list "b"
```

List Trick - Split a list into equal bits

```
a = [1,2,3,4,5,6,7,8,9]
list(zip(*[iter(a)]*3))
# this translates to: make a list of ( create a single tuple from ( 3 iterators of "a
↪" ) )
>>> [(1, 2, 3), (4, 5, 6), (7, 8, 9)]

# iter(a)*3 -> 3 iterators are created with the same ID
# for explanation lets call these iter1.1, iter1.2, iter1.3
# zip(* (iter1.1, iter1.2, iter1.3)) unpacks the iterator with "next"
# (next(iter1.1 pos0), next(iter1.2 pos1), (next(iter1.3 pos3)), (next(iter1.1 pos4),␣
↪...so on
# since the iters are all identical objects, they share the "next" counter
# zip takes the 3 subdivided iters one value at a time and creates a tuple out of 3x␣
↪next calls
# this step repeats until a StopIteration is hit
# the last step is to convert a zip object to a list via: list(zip...)
```

Sets

Sets are the best for storing unique values, finding same value intersects, finding different value intersects or combining unique values. It is far too easy to always use Lists for everything, but always remember that Sets are available to handle unique values very fast and efficiently that would otherwise require more work on a List.

- Syntax: create, add, remove

```
# sets are great to use over lists when the user does not want to keep duplicates
a = {1,2,10}

# to add
a.add(2) # this is duplicate and will not be added
a
>>> {1,2,10}
a.add(4) # this is not a duplicate, therefore it is added
a
>>> {1,2,10,4}

# to remove
a.remove(10)
a
>>>{1,2,4}
```

- find the overlaps between 2 sets

```
a = {1,2,4}
b = {2,3,4}
a.intersection(b)
>>> {2,4}
```

- find the difference between 2 sets

```
#
a = {1,2,4}
b = {2,3,4}
a.difference(b)
>>> {1,3}
```

- get the combined - non duplicate of 2 sets

```
a = {1,2,4}
b = {2,3,4}
a.union(b)
>>> {1,2,3,4}
```

Dictionaries

Dictionaries are great for name-space like structured data (key/value pairs).

Easy to read and use, however it can be tricky to use while writing large pieces of code, since available keys are not auto-completed therefore the programmer has to remember what keys are available for use. For large-code or more-program-friendly use case - classes should be looked at for containing data in its name-space attributes that is auto-completed.

See *List Comprehensions (alt for loops)* for dictionary comprehensions.

```
# syntax
a = {"key1": "value1", "key2": "value2"}
a["key1"] # access value via keys
>>> "value1"

# report out a default value if a key does not exist with "get" instead of raising a␣
↪KeyError
a.get("key3", "not on record")
>>> "not on record"

# add to dict
a = {"key1": "value1"}
a["key2"] = "value2"
# or with "update"
a.update({"key3": "value3"})
# the same syntax can be used to update an existing key/value pair

# iterate through keys and values
for k, v in a.items():
    print(k, v)
>>> "key1 value1"
>>> "key2 value2"
```

Trick - Handling nested dicts

```
# pulling out a sub-dict from sub-dict values
database = {1:{'name': 'bob', 'color': 'blue'},
            2:{'name': 'jay', 'color': 'green'},
            3:{'name': 'kai', 'color': 'blue'},}
# lets pull out a sub-dict database for all color=blue people
subdb = {ID: subdict for ID, subdict in database.items() if subdict['color'] == 'blue
↪'}
```

Trick - Merging 2 dicts (shallow copy)

Note that a shallow copy will create a new dict ID but the key and value objects will still be the same object ID as the originals. (this is only an issue if the original dicts are defined via mutable variables). The example below will not have any issues since strings and integers are immutable.

```
x = {'a': 1, 'b': 2}
y = {'b': 3, 'c': 4}

z = {**x, **y}
>>> {'c': 4, 'a': 1, 'b': 3}
```

Trick - Adding a custom attribute to dict

Ever wish that there was a cleaner attribute to a builtin object? I ran into this when trying to write a user-friendly API but did not want to force the users to necessarily know what a dict.keys() were, instead i wanted to write it out as something much more pythonic

```
# lets say we have a database class that holds IDs of items in a dictionary
class MyDB():
    def __init__(self):
        # we initialize an empty dictionary
        self.items = {}

    # define a method for adding items
    def additem(self,ID,arg):
        # we then update the dictionary storage
        self.items.update({ID: arg})

# now to use it, we could say
db = MyDB()
db.additem(1,'apples')
db.additem(2,'oranges')

# we can display the item IDs by typing
db.items.keys()
# but the results are not very nice to read, nor is db.items.keys() intuitive for␣
↪users
>>> dict_keys([1,2])
```

- wouldnt it be nice if we could just type db.items.ids? Let's see how it's done

```
# same class as before, but self.items = a custom class now that extends dictionaries
class MyDB():
    def __init__(self):
        # we initialize an empty dictionary
        self.items = ExtDict()

    # define a method for adding items
    def additem(self,ID,arg):
        # we then update the dictionary storage
        self.items.update({ID: arg})


# a custom class to extend dictionaries
#  inherit from dict all the builtin attributes of a dictionary
class ExtDict(dict):
    @property
    def ids(self):
        return list(self.keys())


# now let's use it
db = MyDB()
db.additem(1,'apples')
db.additem(2,'oranges')

# lets get the existing IDs a more user-friendly way
db.items.ids    # easy to read/use
>>> [1,2]       # easy to read/use
```

2.2.5 builtin - Context Manager

Context Manager is a function or class that has an "Enter" buildup code, and an "Exit" teardown code that the user is able simply call in a "with" block. The most common example of this is the builtin file IO class "open". Open has an "Enter" that opens file streaming and an "Exit" that safely closes the file for streaming.

Object Oriented Version

- Setup

```
# context manager setup
class MyContextManager():

    # predefine input on "with" code line
    def __init__(self, preload):
        print("context initialized on runtime")
        self.preload = preload

    # enter method is called when the code enters the with block
    def __enter__(self):
        print("inside with block")
        # keep the context manager class simple
        #   by returning another class that does the work from the __enter__ block
        # When a "with MyContextManger(10) mycont:" is called,
        #   we are really saying that mycont = worker_class(10)
```

(continues on next page)

(continued from previous page)

```
        return worker_class(self.preload)

    # exit method is called when the code exits the with block
    def __exit__(self, exc_type,exc_val, exc_tb):
        # exc_type ==> exception type (ie. TypeError, ValueError, KeyError)
        # exc_val  ==> raise exception argument (ie. TypeError("this is the exc_val"))
        # exc_tb   ==> traceback message (where the in the code did the exception␣
↪occur)

        # exit method receives arg containing details of exception raised in the "with
↪" block
        #   if context can handlethe exception, __exit__() should return a true
        #   (ie. error does not need to be propagated)
        #   returning false causes the exception to be re-raised after __exit__()

        # this feature is really nice when trying to graciously handle exceptions
        if exc_type is ValueError:
            print("handled ValueError, raised exception value = ",exc_val)
            return True
        return False
```

- Extend the context manger functionality

```
# define worker class
class worker_class():
    def __init__(self, preload):
        self.preload = preload
    def extended(self, val):
        print(val, self.preload)
```

- How to use it

```
# now to use it
with MyContextManager(10) as mycont:
    # code without any errors
    mycont.extended(100)

>>> "context initialized on runtime"
>>> "inside with block"
>>> 100, 10

# how it works with handled error
with MyContextManager(10) as mycont:
    # code without any errors
    mycont.extended(100)
    raise ValueError("HI!")

>>> "context initialized on runtime"
>>> "inside with block"
>>> 100, 10
>>> "handled ValueError, raised exception value =  HI!"
```

Functional Programming Version

We can accomplish the same task with a function definition as shown above and a library called contextlib

- Define the worker class (same as above, this does not have to be a class object, this example is simply reusing the same code as above for clarity)

```
# define worker class (same as above)
class worker_class():
    def __init__(self, preload):
        self.preload = preload
    def extended(self, val):
        print(val, self.preload)
```

- Setup the context manager function

```
from contextlib import contextmanager

@contextmanager
def MyContextManager(preload):
    # __init__ code goes here
    print("context initialized on runtime")
    try:
        print("inside with block")
        yield worker_class(preload)
    except ValueError as e:
        print("handled ValueError, raised exception value = ",e)
    finally:
        # __exit__ code goes here
        print("__exit__ cleanup code goes here")
```

- Use it

```
with MyContextManager(10) as mycont:
    # code without any errors
    mycont.extended(100)

>>> "context initialized on runtime"
>>> "inside with block"
>>> 100, 10
>>> "__exit__ cleanup code goes here"
```

2.2.6 builtin - Copy True/Shallow/Deep

There are 3 different ways to copy objects in python:

- **True Copy**: ID or old object and new object are the **same**, therefore changing one object also changes the other
- **Shallow Copy**: ID of old object and new object is **different**, therefore the number of contents and immutable contents can be changed by one that will not effect the other, HOWEVER mutable content copied are still linked between the two objects (ie: a = [[1,2],3], copy_a = [[1,2],3] the list [1,2] inside is the same ID, therefore changing the list in "a" will also change the list in "copy_a"
- **Deep Copy**: creates **new** IDs for each object within (very slow!). There is no link between old and new copy

For more description on mutable/immutable see *builtin - Mutable/Immutable*

For examples on copy see *List - Copy*

2.2.7 builtin - Functions

Note: Note that all functions in python end with a return, even when a `return` is not explicitly typed out in your code (of course this is only true if an exception is not raise within the function). When a `return` is not present within a function, the function will simply `return None`.

Note: Note that local variable data will only be stored in memory while the code is within the function loop. Meaning, a variable within a function can only be called within the function. See *Common pitfall - Global vs Local vs NonLocal*

Syntax

```
1  # bare minimum
2  def foo(arg1, arg2):
3      return arg1 + int(arg2)
4
5  # python version 3.5+ added type-hints that improves your editors inline error␣
   ↪handling,
6  #  testing, linters, etc.
7  # to be explicit: everything on line 7 is type-hints. Lines 8-14 is your docstring
8  #  (for when you call help(foo), or autodoc)
9  def foo(arg1: int, arg2: str="0") -> int:
10     """
11     Simple add function
12
13     :param (int) arg1: positional argument one
14     :param (str) arg2: positional argument two
15     :return (int): addition of arg1 + arg2
16     """
17
18     return arg1 + int(arg2)
19
20 # lamba function (no-name function, or inline function definition)
21 lambda arg1, arg2: arg1 + int(arg2)
```

All about variables

- mandatory vs optional arguments
  - see line 7 from syntax example
  - arg1 is mandatory because it has no predefined value
  - arg2 is optional because it has a predefined value of "0", therefore foo can be called by `foo(1)` or `foo(1,"0")` both returning the same value
- positional vs. keyword arguments
  - data must be entered in a positional order, unless it's fed with keyword arguments (see example below)

```
# from the syntax example above with foo():

# call a function with positionals
```

(continues on next page)

(continued from previous page)

```
foo(1,"2")
>>> 3

# call a function by unpacking positionals
mylist = [1, "2"]
foo(*mylist)
>>> 3

# call a function by keywords
foo(arg2="2", arg1=1)
>>> 3

# call a function by unpacking keywords
mydict = {"arg2":"2, "arg1": 1}
foo(**mydict)
>>> 3
```

Common pitfall - Global vs Local vs NonLocal

- General Global variable

```
# global variable
x = 5

def func():
    print(x)

# by default, a function will only be able to access argument variable or variable␣
↪defined
#  within the function
func()
>>> UnboundLocalError: local variable 'x' referenced before assignment

def func(y):
    z = 15
    print(y,z)

# now both variable y and z are accessible, so there are no errors
func(y=10)
>>> 10 15
```

- Accessing a Global Variable

```
# to access a global variable defined outside of the function
def func():
    global x    # make it accessible within the function
    print(x)
    x = 500

# using "global" we can allow our function to reach outside the local variables and
#  grab the variable "x"
x = 5
func()
>>> 5
# notice no error this time, but be careful, the function also altered the value of "x
↪"
```

(continues on next page)

(continued from previous page)

```
# note that "x" in the function is no longer a "local" variable, x is now globally␣
↪redefined!
x
>>> 500
```

- Accessing a Nonlocal Variable (nested functions or loops within a function)

```
# alternatively we can access variables from nested functions via "nonlocal"
def func():
    y = 5
    def func2():
        nonlocal y
        print(y)

func()
>>> 5
```

Call function by its string name

- Call a function by string when it is imported with `getattr()`

```
import file1

# in this case: getattr(module, a function is a property of a module)(arguments for␣
↪function)
getattr(file1 , "foo")(1,"2")
>>> 3
```

- Call a function by string in the same file with `globals() locals()`

```
# suppose we have a simple add function:
def func(a,b):
    return a + b

# locals and globals will return the same here
locals()["func"](1,2)
>>> 3
globals()["func"](1,2)
>>> 3

# the difference between locals and globals comes in when a property is nested
def nest():
    def func(a,b):
        return a - b
    # locals will look in the current layer (ie. within nest())
    print("from local: ", locals()["func"](1,2))
    # globals will look at the module layer
    print("from global: ", globals()["func"](1,2))

nest()
>>> 'from local: -1'
>>> 'from global: 3'
```

Function dundur

```
# string name of a function
foo.__name__
>>> 'foo'

# list of arguments of a function
foo.__code__.co_varnames
>>> ('arg1', 'arg2')
```

functional programming: map, filter, and reduce

```
# map works-on multiple iterables at the same time
#   take the following 2 lists and a simple add function for instance
a = [1,2,3]
b = [4,5,6]
def add(x,y):
    return x + y
list(map(add, a, b))
>>> [5, 7, 9] # 1+4, 2+5, 3+6

# use filter to narrow down a iterable with a custom true/false function
a = [1,2,3]
def test_odd(x):
    return x % 2 # returns 0 for even (same as true), 1 for odd (same as false)
list(filter(test_odd, a))
>>> [1, 3]

# use reduce to narrow down a iterable to a single value
a = [1,2,3]
def multiply(x,y):
    return x*y
reduce(multiply, a)
>>> 6
```

functional programming - factory/closures/currying

Factory - A function that keeps its own internal state (see example below) Closure - A "Factory" assigned to a variable Currying - Similar to a "Closure" but input arguments changes the functionality of the Closure

- Simple Function (NOT A FACTORY example to stage set)

```
# the following function is not a "factory" because its state is not internal
#   meaning, that each instance of a function will carry the same state, see demo␣
↪below:
counter = 0 # some global initial state
def incrementer():
    global counter # allow access to global variable "counter" within the function
    counter += 1 # increment the "state", but note that counter is linked to a global␣
↪state
    return counter

incro1 = incrementer() # if incrementer was a "factory", incro1 would be called a
↪"Closure"
```

(continues on next page)

(continued from previous page)

```
incro2 = incrementer()

# it doesnt matter that we have 2 instances of the function,
#   their "state" is linked to a global variable
incro1 # incro1 was assigned as a function, therefore calling the function doesnt␣
↪require another "()"
>>> 1

# we would expect that incro2 would also return 1 but their "state" is linked
incro2
>>> 2
```

- Factory Function

```
# this is what makes factories unique from regular functions,
#   their internal state unique to each instance
def incrementer():
    counter = 0 # internal state set
    def return_func():
        nonlocal counter # allow access to one level higher variable; ie. "counter"
        counter += 1 # change the internal state
        return counter # this is whats ultimately returned when called by a "Closure"
    # this is the tricky part...
    # when incrementer is initially defined, it runs through the code inside the␣
↪function
    # sets initial "counter" state to 0
    # setups up a function "return_func()" but does nothing with it
    # then! the "Closure" variable is actually == the "return_func"
    # this is why counter = 0 is never reset after initialized, because
    # calling the "Closure" variable is actually calling "return_func"
    return return_func

# lets see this in practice...
incro1 = incrementer() # incro1 is a "Closure" that sets counter = 0 and returns␣
↪incro1=return_func
incro2 = incrementer() # lets make a second copy to demonstrate that "state" is unique

# note that in this case we have to put "()" since incro=return_func,
#   and to call return_func we need: return_func()
incro1()
>>> 1
incro2()
>>> 1
incro2()
>>> 2
incro2()
>>> 3
incro1()
>>> 2 # indeed state is unique to each instance!
```

- Currying Function (special Closure)

```
# now is the best time to show a builtin - library shortcut from "functools" called
↪"partial"
#   "partial" creates a "Closure" for you
from functools import partial
```

(continues on next page)

(continued from previous page)

```
def multiply(x, n=1):
    return x * n

# times3 is a unique "Closure" created from a "Factory" of multiply
times3 = partial(multiply, n=3)
# another unique "Closure" built from the same "Factory" but with different function
times5 = partial(multiply, n=5)

times3(2)
>>> 6
times5(2)
>>> 10
```

Trick - Clean Function Piping

Ever need to rip through a bunch of "if" statements to call the function you want? Try combing a piping dictionary with function calls.

```
def func_one(a,b):
    return a+b

def func_two(a,b):
    return a-b

def func_three(a,b):
    return a*b


def math(val1: float=0.0, val2: float=0.0, condition: str="one") -> float:
    """
    Takes a value and multiplies it by a string amount

    :param (float) val: input value
    :param (str) multi: multiplier in string
    :return (float): multiplied input value
    """

    piper = {"one": func_one,
             "two": func_two,
             "three": func_three,}

    try:
        # instead of coding up a bunch of if condition == something, you can make use␣
↪of a
        #  dict's keyword arguments to pipe for you
        return piper[condition](val1, val2)
    except KeyError:
        raise UserWarning(f"Incorrect input value for condition={condition}")
```

Trick - Define function via text

```
# use exec() to execute text and add it to the global variables
exec("def f(x): return x*2", globals())
```

(continues on next page)

(continued from previous page)

```
f(5)
>>> 10
```

2.2.8 builtin - File Input/Output (IO)

Syntax

Although there are many ways to work with files, by far the safest is with "Context Manager", see *builtin - Context Manager*. A Context Manager has an Enter and an Exit protocol that handles events. In terms of file IO, `with open` opens a file stream on Enter, and closes file stream on Exit, therefore files are always closed down once the code is done streaming it.

```
# read a file
with open("filename.txt", "r") as f:
    while True:
        line = f.readline()
        if not line:
            break

# write a file
text = ["hello", "world"]
with open("filename.txt", "w") as f:
    for line in text:
        f.write(line + "\n")
```

Open file object modes:

- `r`: read only
- `w`: write only
- `x`: execution only
- `a`: append to existing file
- `b`: binary mode (this is combined with `r` or `w`
- `+`: open for updating (reading/writing)

Working with directories

2.2.9 builtin - Logic Loops

- There are infinite ways to write code but it is important to highlight the wise words from The Zen of Python: "Flat is better than nested" in this section more than any other because it is far too easy to nest logic loops that are confusing to read after numerous indents. As a suggestion try to limit nested loops to 3-4; not only will it help downstream users pickup your code easier but the code will also be easier to test. Logic that needs more than 3-4 nested loops should be broken up into separate functions.

True False

In python the following all evaluate to True:

- 0 == True (note that float 0.0 is not True, only the int 0)

- 'any non empty str' == True
- non empty set, tuple, list, dict

General "and" "or" "not" "any" "all"

```
2 == 5 or 10 == 10
>>> True

2 == 5 and 10 == 10
>>> False

2 != 5 and 10 == 10
>>> True
# in a alternate form
not 2 == 5 and 10 == 10
>>> True

# "any" is powerful with list comprehensions:
any(i == 4 for i in [3,4,5])
>>> True
any(i == 10 for i in [3,4,5])
>>> False

# "all" works similar to "any" but all instances of the iterable must eval to True
all(i%2 == 0 for i in [4,6,8])
>>> True # because all num/2 result with a remainder of 0 (the values are all even)
all(i%2 == 0 for i in [4,5,8])
>>> False # this evals to True, False, True which is overall False

# there is one more that can be imported from builtin library "operator"
from operator import xor
# xor: one or the other true but not both
1 == 1 xor 2 == 1
>>> True
```

if elif else

```
x = 5
if x == 5:
    print('x == 5')
elif x == 6:
    print('x != 5 but x does equal 6!')
elif x == 7:
    print('x !=5, x != 6, but x == 7')
else:
    print('x was not equal to 5, 6, or 7')

# there is a 1 liner short for a simple if/else
y = x + 10 if x == 5 else 0
>>> y = 15 # these are really useful for initializing variable
```

for loop

```
# iterate through strings by char
for char in "this":
    print(char)
>>> 't'
>>> 'h'
>>> 'i'
>>> 's'

for value in range(start=1, stop=30, step=10):
    print(value)
>>> '1'
>>> '11'
>>> '21'

# iterate through sets, tuples, lists
for value in [10,20,30]:
    print(value)
>>> '10'
>>> '20'
>>> '30'

# it is often useful to iterate through the values and also keep index
for index, value in enumerate([10,20,30], start=100):
    print(index, value)
>>> '100 10'
>>> '101 20'
>>> '102 30'

# iterate through dicts (iterate on keys, values, or items)
for key, value in {'key1':1, 'key2':2}.items():
    print(key, value)
>>> 'key1 1'
>>> 'key2 2'

# for loop on multiple same same iterators
for val1, val2 in zip([1,2,3],[10,20,30]):
    print(val1,val2)
>>> '1 10'
>>> '2 20'
>>> '3 30'

# use break to jump out of a for loop early
for val in [1,2,3]:
    if val == 2:
        break
    print(val)
>>> '1'
# but never gets to printing 2 or 3

# use continue to jump ahead of the current iteration (same as a __next__() call)
for val in [1,2,3]:
    if val == 2:
        continue
    print(val)
>>> '1'
```

(continues on next page)

(continued from previous page)

```
>>> '3'
# note how 2 was skipped
```

List Comprehensions (alt for loops)

```
# a simple for loop
vals = []
for value in colletion:
    if condition:
        vals.append(expression)

# can be written in 1 line with list comprehension
vals = [expression for value in collection if condition]
```

- example

```
vals = []
for value in [1,2,3]:
    if value%2 == 1:
        vals.append(value + 10)

vals
>>> [11,13]


# now with list comprehension
vals = [value + 10 for value in [1,2,3] if value%2 == 1]

vals
>>> [11,13]
```

- similarly dictionaries can also be handled with list comprehensions

```
vals = ["/".join(key, str(value)) for key, value in {'one': 1, 'two': 2}.items()]

vals
>>> ['one/1', 'two/2']
```

- dictionary comprehensions

```
vals = {k: 2*v for k, v in {'one': 1, 'two': 2}.items()}

vals
>>> {'one': 2, 'two': 4}
```

while loop

```
i == 0
while i < 3:
    print(i)
    i += 1
else:
    'while loop finished without a break'
```

(continues on next page)

(continued from previous page)

```
>>> '1'
>>> '2'
>>> '3'
>>> 'while loop finished without a break'

i == 0
while i < 3:
    print(i)
    if i == 2:
        print('while loop finished early with a break')
        break
    i += 1
else:
    'while loop finished without a break'
>>> '1'
>>> '2'
>>> 'while loop finished early with a break'
```

try/except/pass

See full list of exception at Link

```
try:
    # somecode to test for exceptions
except NameError:
    # somecode raised a NameError, do something
except (ValueError,KeyError):
    # samecode did not raise a NameError, but it did raise either
    #   a ValueError or KeyError, do something
except:
    # catch all other errors, this is lazy coding - try to not use this
    #   the owner should understand what exceptions occur and handle it appropriately
else:
    # no exception were raised, do something
finally:
    # run code lastly before exiting try loop, no matter if an exception was or not
```

Trick - Type check multiple variables at the same time

```
# "all" is a check that all argument are True inside
# isinstance checks if an object is made of some parent object (ie, isinstance("this",
↪ str) == True)
# the rest is a list comprehension to cycle through multiple objects on the same line
if not all(isinstance(variable, (int, float) for variable in [a, b, c, d])):
    raise TypeError("Incorrect variable type entry, a,b,c,d must be either int or␣
↪float")
```

2.2.10 builtin - Mutable/Immutable

- Mutable: An object that can be changed after it is created
  - list, set, dict

```
a = [1,2,3]
id(a)
>>> 63674848
a.append(4)
id(a)
>>> 63674848 # note that the ID stays the same, however the content of the list␣
↪changed
```

- Immutable: An object that cannot be changed after it is created
  - bool, int, float, str, tuple, frozenset

```
# same ID every time you call id on a bool, int, float, str
id(123)
>>> 1794170960
a = (1,2,3)
id(a)
>>> 69202848 # now note once you change the tuple size the ID rolls
a += (4,)
id(a)
>>> 65455520 # new id
```

2.2.11 builtin - Packaging (managing python imports/files/projects)

Managing imports can be quite a challenge in python at first. There are builtin python packages, packages you install (through pip), and your own files that can all be imported into your project. This section will hopefully shine some light on how imports work in python, and give you an idea on how to setup your own package/library. Note that folks use library and package in python interchangeably. We will use "packages" since that is how python folder system calls them.

Level Setting Imports

Importing syntax and meaning

1.) Import everything under a package (this can be slow and not preferable since it is not explicit)

```
import unittest

# now use it via
mycase = unittest.TestCase

# we can also use our own custom shortname for an import via:
import unittest as ut

# now use it via
mycase = ut.TestCase
```

2.) Import specifics (preferred option because it is explicit)

```
from unittest import TestCase

# now use it via
mycase = TestCase
```

Importing Global Site-Packages

Site-packages are accessible to the python interpreter no matter where your current working directory may be (see in more detail *How Do Site-packages (builtin or pip installed packages) Import*).

```
import unittest # under Python/Lib as a module folder
import csv # under Python/Lib as a module file
import pandas # under Python/Lib/site-packages as a module folder (if pandas is pip␣
↪installed)
```

Importing your own files in a working directory.

```
Folder structure:

    folder1
    |-file1.py
    |-file2.py
    |-folder2
        |-file3.py
```

1.) files in the same folder as your original script

```
# this code is in file1.py

# we dont need to type ".py" at the end of the file names
import file2

# to use a function "foo" that is in file2.py
file2.foo()
```

2.) import files from a sub-directory

```
# this code is in file1.py

# sub-folders are handled by "." instead of "/" in python imports
import folder2.file3

# to use a function "foo" that is in file3
folder2.file3.foo()
```

3.) import files from a directory above

Note: You can only import files on the same directory level as your starting script file! Never higher! You will get an `ValueError: attempted relative import beyond top-level package`

```
# this code is in file3.py

# the following works if we launch our original script: python file1.py
#  in file1.py the same code exists as in bullet 2) that then calls file3.py
# note: this works because file1.py (top level directory file was launched)
#  if you tried to run file3.py by itself: python file3.py it will fail because of␣
↪the note above
from file2 import foo
```

(continues on next page)

(continued from previous page)

```
foo()
```

How to make your package accessible to your python instances

Let's say we wrote a piece of python code that we want to reuse. We have a few very annoying options to directly call this python file from any directory:

- we can brute force copy the file from one directory to the next so we that we can locally import it (terrible option, never do this)
- we can absolute/relative path import it into our other project (not good solution since folder paths change all the time)
- we can add our python script location to our system environment PATH. This can be done by either editing our windows account environment PATHs or within python using `sys.path`. (now we are getting warmer but this solution still depends on file paths that again might change)

Note: The best solution is to add your python file/package to your `Python/Lib/site-packages` folder. All site-package scripts and packages are available to any python instance you may spin up and importing is just as easy as any other site-package import (ex: `import mycustompackage`)

How Do Site-packages (builtin or pip installed packages) Import

Python comes with a few very handy builtin packages all stored in the you `Python/Lib` like os, csv, html, etc., and `Python/Lib/site-packages` for pip installed packages like pandas, pyqt and so on. These packages are directly accessible to any python interpreter as long as it is in `Python/Lib` or in `Python/Lib/site-packages`. Let's take a look at how an existing one works:

1.) A python package stored in `Python/Lib` or `Python/Lib/site-packages` is accessible to your python interpreter no matter where the interpreter was launched from!

2.) Take for instance the `unittest` builtin package. It is located under `Python/Lib` Pay special attention to the fact that `unittest` is actually a folder.

3.) We can import this package from any python interpreter by typing

```
import unittest
```

4.) The code above import all of unittest. So that great and all but how does it work? `unittest` as we noted earlier is a folder. Well when we python treats folders like modules so long as that folder contains a `__init__.py`. Open up `Python/Lib/unittest` and convince yourself that, that is in fact true.

5.) Now we are faced with another question. What in the world is a `__init__.py` file? A `__init__.py` is referred to as a constructor, it converts a folder into a module that can be imported by python, and when imported all functions/classes within `__init__.py` are imported (in this case `unittest.TestCase` for instance)

```
# sample code from: Python/Lib/unittest/__init__.py

__all__ = ['TestResult', 'TestCase',]

from .result import TestResult
from .case import TestCase
```

```
# our script can use unittest and it's imported results by...
import unittest

# this works because python imported unittest from Python/Lib
#  then unittest/__init__.py imported under the hood TestResult and TestCase
mytestcase = unittest.TestCase
```

```
# note that we could just as well have jumped straight to TestCase if
#  that is all we were using (this is always more preferred to import only what you need)
from unittest import TestCase

# now use it by...
mytestcase = TestCase
```

6.) Before closing out this site-package example, let's take a look at `__all__`.

6.1) By default python a general import call: `import unittest` will import all functions/classes/modules that are listed in the `__init__.py` file.

6.2) `__all__` will prevent the user from importing anything that is not explicitly stated in the `__all__ = [...]` list. Note, this is only true if the user uses the explicit import form `from unittest import *`.

Note: `__all__` does not exempt the user from directly importing a hidden function. For example lets suppose there is a function under unittest called `hiddenfunc` we could bypass the `__all__` restriction by directly importing the function name `from unittest import hiddenfunc` or simply just using the general import `import unittest` that import everything in the `__init__.py`

How to structure your own package

more on __init__ and __all__

2.2.12 builtin - Strings

Syntax

```
# strings can be represented in 2 different ways:
"string with double quotes"
'string with single quates'

# benefit for double quotes: ability to use the apostrophe (')
"it's a nice day"

# string literals (interpret string as a string without special character interpretation)
"this is with a new line\n"
r"this is without a new line\n"
# note that the "r" signifies string literal and the "\n" will not be interpreted as a LFCR

# binary string
b"this is binary"
```

```
# multi-line string
'''
This
is
on
multi-lines
'''
>>> '\nThis\nis\non\nmulti-lines\n'

# Careful! This is not the same!
'''This
   is
   on
   multi-lines'''
>>> 'This\n   is\n   on\n   multi-lines'
```

Common String Tools

Note: examples below assume the variable "x" is a string variable (ex: x="thiss")

- To uppercase: `"this".upper()` >> "THIS"
- To lowercase: `"ThiS".lower()` " "this"
- To title: `"this is a title".title()` >>> "This Is A Title"
- To add to string: `x.append("that")` or `x += "that"`
- To count char occurrence: `x.count("s")` >>> 2 because x="thiss" and there are 2x "s"
- To find the index of a char: `x.index("h")` >>> 1
  - note string index start from 0
  - index will return the first occurrence from the left
  - similarly you can use `x.find("s")` >>> 3 or `x.rfind("s")` >>> 4
- To slice a string: variable[start:end:reverse]
  - slicing start will include the starting char, but NOT the ending char `"this"[1:3]` >>> "hi"
  - start/end field can be left blank to grab all of the start/end `"this"[1:]` >>> "his"
  - reverse order `"this"[::-1]` >>> "siht"
- To replace a char(s): `"this this this".replace("thi", "set", 2)` >>> "sets sets this"
- To check if str is a int: `"345".isdigit()` " >>> True
- To check if str is numeric: `"345.1".isnumeric()` " >>> True
- To find the length of a str: `len("this")` >>> 4 (subtract 1 if len of a string is used for indexing)
- To sort a str: `sorted("this")` >>> ['h', 'i', 's', 't']
- To join strings: `"/".join(['this','that'])` >> "this/that"
- To split strings: `"this and that".split(" ")` >>> ['this', 'and', 'that']
- To add a unix formatted new line (line feed): `"this\n"`

- To add windows carriage return + line feed: `"this\r\n"`
- To add a tab: `"this\t"`
- To check if a string contains all digits: `"1234".isdigit()` >>> True, Note however `"-1234".isdigit()` >>> False because the negative sign is not a digit.
- To check what a string start with or ends with: `"this".startswith("t")` >>> True

String Arguments and Formatting

```
# f-strings (python3+)
f"x is equal to {x}"
# benefit is that f-strings allows you to perform arithmetic/logic on the spot
f"x is equal to {x + 5}"
f"x is equal to {x if x < 5 else x + 5}"

# format (python2-3)
"x is equal to {}".format(x)
"x is equal to {x1}".format(x1=x)

# % "modulo operator"
"x is equal to %(x1)d" % {"x1": x}
```

Formatting the argument injections

- {:5.2} 5 in this case is the str-length,and 2 is number of significant digits note significant digits overrule: {:3.5} will have a str-len of 6 chars for a positive number (5 digits and a ".") {:3.5} will have a str-len of 7 chars for a negative number (5 digits a "-" and ".")

```
f"{1:4}"
>>> '   1'
f"{1.11111:4}"
>>> '1.11111' # not what you would expect str-len is not 4
f"{1.11111:4.2}"
>>> ' 1.1'
f"{1.11111:2.4}"
>>> '1.111' # note that sigfig wins vs str-len
```

- < > = ^: left, right, padding of characters, center rules

```
f"{1:<4}"
>>> '1   '
f"{1:>4}"
>>> '   1'
f"{1:0=4}"
# note padding only works on int or float
>>> '0001'
f"{1:^4}"
>>> ' 1  '
```

- "+" "-" "space": use sign for both pos/neg values (ie: "+5" and "-5"), sign for neg only ("5" "-5"), use sign for neg only but leave space for positive (" 5" "-5")

```
f"{1:+}|{-1:+}|{1:-}|{-1:-}|{1: }|{-1: }"
>>> '+1|-1|1|-1| 1|-1|'
```

- d: int

```
f"{123:d}"
>>> '123' # note that this does not convert a float to a int or str to int
```

- f: float (by default 6 decimals)

```
f"{1:f}"
>>> '1.000000' # note flag f does convert a int to a float but NOT str->float
```

- e and E: exponent with small "e" or large "E" (default 6 decimals)

```
f"{1:e}"
>>> '1.000000e+00' # similar to float conversion
```

- g: The precise rules are as follows: suppose that the result formatted with presentation type 'e' and precision p-1 would have exponent exp. Then if -4 <= exp < p, the number is formatted with presentation type 'f' and precision p-1-exp. Otherwise, the number is formatted with presentation type 'e' and precision p-1. In both cases insignificant trailing zeros are removed from the significand, and the decimal point is also removed if there are no remaining digits following it. Positive and negative infinity, positive and negative zero, and nans, are formatted as inf, -inf, 0, -0 and nan respectively, regardless of the precision. A precision of 0 is treated as equivalent to a precision of 1. The default precision is 6.
- %: percentage. Multiplies the value by 100 and uses (f) format followed by a percent sign

```
f"{1:%}"
>>> '100.000000%' # similar to float conversion
```

- ,: to separate every 1000 by a comma

```
f"{1000:,}"
>>> '1,000'
```

- positional arg call:

```
"pos0={0}, pos2={2}, pos0={0}".format(*[10,20,30])
>>> 'pos0=10, pos2=30, pos0=10'
```

Trick - Replace multiple spaces with a single space

```
text = "this   is   spaced odd       but it's     okay"
" ".join(text.split())
>>> "this is spaced odd but it's okay"
```

Trick - Hide print statements (closure)

```
import os, sys

class HiddenPrints:
    def __enter__(self):
        # log the original stdout setting
        self._original_stdout = sys.stdout
        # buffer stdout into an empty path
        sys.stdout = open(os.devnull, "w")
```

(continues on next page)

(continued from previous page)

```
    def __exit__(self, exc_type, exc_val, exc_tb):
        # close the buffer
        sys.stdout.close()
        # reset stdout setting to original
        sys.stdout = self._original_stdout
```

2.2.13 builtin - User Inputs

Runtime Inputs

```
# python3: runtime input (stops your code and waits for a user input)
store = input("Text asking for input: ")
```

- python hack for "switch" statement

```
# python doesnt have a "switch" keyword function builtin, but you are given the power␣
↪to forge one!
#   lets say you have multiple options on input
print("Options:\n"
      "1) Main Menu\n"
      "2) Names\n"
      "3) Addresses\n")
# functions that handle each menu:
def main_menu():
    pass
def names():
    pass
def addresses():
    pass
# switch statement to pipe the inputs
piper = {"1": main_menu,
         "2": names,
         "3": addresses}
# the dict stores the function objects, and piper[input]() then calls the function␣
↪upon valid input
piper.get([input("Enter option 1, 2, or 3: ")], "Invalid Input")()
```

Commandline Inputs

```
# ask for user inputs when calling the script
#   note that sys.argv[0] is the name of the script (in this example "script.py")

# access argument via sys
import sys

# we are looking for 3 inputs, else give guidance for proper input format
if len(sys.argv) == (1+3):
    name_first = sys.argv[1]
    name_last = sys.argv[2]
    age = sys.argv[3]
else:
    print("Script Usage:\n"
```

(continues on next page)

(continued from previous page)

```
        "script.py arg1 arg2 arg3\n"
        "where,\n'
        "arg1 == First name\n"
        "arg2 == Last name\n"
        "arg3 == age\n")
```

2.2.14 builtin - pdb (python debugger)

PDB is python native debugger, it comes with every python installation therefore it is reliable across all platforms (wheather you are on a coworkers PC or ssh'ed into a remote server).

- execute your code via pdb: `python -m pdb file.py`
- insert a breakpoint into your script via `import pdb` then `pdb.set_trace()` above the line you would like to break at
- jump to next line `n`
- jump to next breakpoint `x`

2.2.15 lib - Django (web framework)

Django is one of many web frameworks out there available for use by python developers. Django is by far one of the most popular frameworks out there because it is easy to use yet it has a lot of depth to it (very similar to the python language itself).

Quick Setup Guide

1.) Create a git repo and clone it down (see *tool - Git*)

2.) Create a venv within the repo and activate the venv (see *lib - virtualenv*)

3.) Install required packages: `pip install django`

4.) Setup Django Project Structure: `django-admin startproject projectname`

5.) Restructure Folder: Django creates a sub-folder within a "projectname" this is a bit redundant, here is an easier setup:

```
git_repo
   |
   |-projectname
   |   |
   |   |-__init__.py
   |   |-settings.py
   |   |-urls.py
   |   |-wsgi.py
   |-manage.py
```

6.) Check that the initial setup was successful, locally launch the site: `python manage.py runserver`

7.) Create a new app `python manage.py startapp appname` NOTE: you have to do this step (step 7) before step 8. You will get a "ModuleNotFoundError" if you try to run a `startapp` that is already part of the `INSTALLED_APPS` list.

7.1) `admin.py` is your django admin settings

7.2) `apps.py` is your app settings

7.3) `models.py` is your database tables

7.4) `test.py` test cases

7.5) `views.py` settings for what will be displayed as HTML

8.) After a new app is created it must be added to the `settings.py` > `INSTALLED_APPS` list It is the directory name of the app that is to be added to the apps list

```
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'appname'
]
```

9.) Create a `templates` folder in the app folder and create a html file for the app

```
<body>Hello world!</body>
```

10.) Create a `urls.py` inside the app folder that django will use to look for any URLs that belong to the app

```
# general django path function
from django.urls import path

# your custom URLs (views.py is already created when the app was setup)
from appname import views

# it is a good idea to give each url a "name=" for easier debugging down the road
urlpatterns = [
    path('',views.hello, name='hello_world'),
]
```

11.) Create the corresponding function view inside the `views.py` that pipes the function name to the html file

```
from django.shortcuts import render

# Create your views here. The render method will look for html files within a
↪"templates" directory
def hello(request):
    # context is data that we can pass to an HTML template
    context = {}
    return render(request, 'hello_world.html', context)
```

12.) Finally add the path the `app/urls.py` into the project `urls.py`

- `path('',` is the home landing page, the same was as `path('admin/'` is the landing page for `yoursite/admin`
- `include('hello_world.urls')` is which app urls should be piped when the user lands the page

```
from django.contrib import admin
from django.urls import path
```

(continues on next page)

(continued from previous page)

```
# to hook up your custom URLs
from django.urls import include

urlpatterns = [
    path('admin/', admin.site.urls),
    path('', include('hello_world.urls'))
]
```

13.) check that the initial setup was successful `python manage.py runserver`

How to Pass Variables to HTML (via context)

The templating code that Django uses within `HTML` is very similar to `JINJA2` (see more at Django Docs. We can access data we passed through our `context dictionary` like so:

- Let's say our context dictionary contains the following (inside our `views.py`):

```
# this is within our "views.py" file

def hello(request):
    data = {
        "var1": 1,
        "var2": [10,20,30],
        "var3": {"var3key1": "value1", "var3key2": "value2"}
    }
    data2 = {
        "var1": "data from data2 variable"
    }
    # context is data that we can pass to an HTML template
    context = {
        "data_key": data,
        "data2_key": data2,
    }
    return render(request, 'hello_world.html', context)
```

- To access a variables in HTML. Note each `key` in `context` is a direct variable that can be accessed

in the HTML. For demonstration on what name gets used where, the following example will have slight different names for keys/variables (we would not do this in practice, for simplicity name keys/variables the same).

```
<body>
    <p>
        Here is how we call a variable {{ data_key.var1 }} <br>
        Here is how we call a variable within a list {{ data_key.var2.0 }} <br>
        Here is how we call a variable within a dict {{ data_key.var3.varkey1 }} <br>
        Here is how we call another variable {{ data2_key.var1 }} <br>
    </p>
</body>
```

- For Loops with variables

```
<body>
    <p>
        Here are items from a list:
        {% for item in data_key.var2 %}
```

(continues on next page)

(continued from previous page)

```
            <li>{{ item }}</li>
        {% endfor %}
    </p>
</body>
```

- If, elif, else

```
<body>
    <p>
        {% if item data_key.var1 == 1 %}
            variable is equal to 1
        {% elif == 2 %}
            variable is equal to 2
        {% else %}
            variable is not equal to anything
        {% endif %}
    </p>
</body>
```

How to Pass Variables via URL address

Passing variables as URL address names has many different use cases. One such case can be template reuse. Image creating a HTML template that displays data to the user, like a blog post. A we don't want to create a copy of that HTML template for each blog post we write, but we also dont want to just overwrite the same template with new blog post data (because what if we want to link a blog post to a friend, we need a permanent URL link). How do we go about achieving such structure? The answer is URL variables:

1.) Setup the app's `url.py` file with the address as a variable name:

```
# file: url.py within the app folder

# general django path function
from django.urls import path

# your custom URLs (views.py is already created when the app was setup)
from projects import views

# here we define our website to take the address after the index site name as a␣
↪variable "pk"
#   localhost:8000/blog/1 would mean pk=1
#   localhost:8000/blog/2 would mean pk=2
# note that the declaration format is "<type:VariableName>/"
urlpatterns = [
path('',views.blog_index, name='blog_index'),
path('<int:pk>/',views.blog_detail, name='blog_post'),
]
```

2.) Setup the corresponding views method. Note that `blog_detail` take a argument `pk` that corresponds to the `url.py` URL variable name

```
def blog_index(request):
    posts = Posts.objects.all()
    context = {
        'posts': posts
    }
```

(continues on next page)

(continued from previous page)

```
    return render(request, 'blog_index.html', context)

def blog_detail(request, pk):
    # now we can query the database for the post's contents and pass that to our HTML␣
↪for rendering
    post = Posts.objects.get(pk=pk)
    context = {
        'post': post
    }
    return render(request, 'blog_post.html', context)
```

Setting Up Project Wide Templates

To get the same formatting on all your app pages. Anywhere from same layout, fonts, colors etc.

1.) Create a `templates` folder within the project folder and create a `base.html` file within it that contains any HTML that is the same for all templates

```
<html>
<head>
    <title> This is a constant title across all html templates </title>
</head>

{% block page_content %}{% endblock %}

</html>
```

2.) Add the templates HTML file to your django project `settings.py`

```
TEMPLATES = [
    {
        "BACKEND": "django.template.backends.django.DjangoTemplates",
        "DIRS": ["personal_portfolio/templates/"],
        "APP_DIRS": True,
        "OPTIONS": {
            "context_processors": [
                "django.template.context_processors.debug",
                "django.template.context_processors.request",
                "django.contrib.auth.context_processors.auth",
                "django.contrib.messages.context_processors.messages",
            ]
        },
    }
]
```

3.) Decorate your app HTML files with the template HTML format

```
{% extends "base.html" %}

{% block page_content %}
<h1>Hello World!</h1>
{% endblock %}
```

Setting Up Static Files

Django uses a folder name `static` to look for any static files you may have within your project. These can be CSS code blocks used for your `base.html`, images, etc. Here are the steps in setting it up:

1.) Create a `static` folder within your project and for this example let's create a subfolder images with a image file

```
git_repo
   |
   |-projectname
   |   |
   |   |-templates
   |   |   |-base.html
   |   |-__init__.py
   |   |-settings.py
   |   |-urls.py
   |   |-wsgi.py
   |-manage.py
   |-static
      |-images
         |-myimage.png
```

2.) Add the static file path to your `setting.py` file

```
STATIC_URL = '/static/'

STATICFILES_DIRS = [
    os.path.join(BASE_DIR, 'static'),
]
```

3.) Start using the static files within your HTML. The following file is our `base.html` but the same rules applies to all other html files you want to use static files in

```
{% load static %}

<img src="{% static 'image/myimage.png' %}">
```

Setting Up Admin

Django sets up a lot of really nice boiler plate website/user/group and more editing via the Admin site. In order to log into the `localhost/admin` site we need the step through the following:

1.) If `makemigrations` has not been run to setup a database where users can be stored:

```
python manage.py makemigrations
```

2.) Makemigrations only prepares the the necessary settings for django, to implement them we need to:

```
python manage.py migrate
```

3.) Now that the database is setup we can create a superuser

```
python manage.py createsuperuser
```

4.) To use the superuser, run the server `python manage.py runserver` and navigate to `localhost/admin/` and log in with your user ID and password

Setting Up Databases

To store any kind of data on your website you have to go through a database. Django uses Structured Query Language (SQL) under the good as it's database language, however Django has written a Object Relational Mapper (ORM) that wraps the whole database experience in python code only.

1.) Database interfaces are unique to each app, and the ORM interface in located in the `model.py` file Here is a list of field types for django: Field Types

```
from django.db import models

class BlogData(models.Model):
    title = models.CharField(max_length=100)
    desc = models.TextField()
    group = models.CharField(max_length=20)
    img = models.FilePathField(path="/img")
```

2.) Using Django's to restructure your `model.py` classes into the format it needs to write our SQL code of the data structure you specified in python code. Note if you get an error: `No installed app with label 'appname'.` then you need to add your app to the project TEMPLATES list in the `settings.py`

- To setup migrations (note you will also need to migrate after this command): `python manage.py makemigarations appname`
- All subsequent migrations: `python manage.py migrate appname`

3.) How to add data to our database from django shell

3.1) Start Django shell: `python manage.py shell`

3.2) Import your Database Class and edit/save

```
# where "blog" is the app folder name that has a "models.py" where we␣
↪defined our "BlogData"
from blog.models import BlogData

# create a instance of our Class BlogData
post1 = BlogData(
    title = 'first title',
    desc = 'first description'
    group = 'group1'
    img = 'img/pic1.png'
)
# save it to the database class
post1.save()

post2 = BlogData(
    title = 'second title',
    desc = 'second description'
    group = 'group2'
    img = 'img/pic2.png'
)
post2.save()
```

4.) How to access data from our database (be sure to `python manage.py migrate`) before trying to access data that you just saved from step 3. The full list of database querys are listed here: Django query / Django filters

```
# to access the database we first need to import it
from blog.models import BlogData
```

(continues on next page)

(continued from previous page)

```
# then we can get all items stored
all_posts = BlogData.objects.all()

# query just a single post by primarykey (pk)
post = BlogData.objects.get(pk=1)

# query by any other file name that we specified
post = BlogData.objects.get(title='first title')

# access data from the post
post.title
>>> 'first title'
post.pk # this is the primarykey
>>> 1
post.id # this is the primarykey as well
```

5.) How to add database models to the admin page by adding it to the app's `admin.py`:

```
# import the models
from blog.models import BlogData


# create a dummy django class for it
class AdminBlogData(admin.ModelAdmin):
    pass

# add it to the admin page
admin.site.register(BlogData, AdminBlogData)
```

2.2.16 lib - pyinstaller

pyinstaller is a packaging module for python projects. It is most efficiently used with virtualenv where only the python packages needed for the project are packaged up (thereby creating a smaller .exe).

Installation

```
pip install pyinstaller
```

Usage

1) create a virtualenv and activate it
2) install necessary python packages for the project
3) complete the project
4) run pyinstaller
    1) pyinstaller does not need to be locally installed within your project for it package up your project, however it does help to keep a stable working version locked with your project. If your global python path is setup (you are able to type `python` in your terminal), then `pyinstaller` should also work if it is installed. Normally pyinstaller will be in the Scripts folder

2) running pyinstaller: `pyinstaller yourmodule.py --onefile`

3) if everything goes well this will create 3 items:

- build folder: contains compiled parts of your package and log files
- dist folder: contains your single .exe file
- .spec file: instructions for pyinstaller to create an executable

TroubleShooting

1) always check your package runs before compiling
2) build the project and see what the error log says when running the .exe it is usually an import error or version incompatability.

- Fixing import error. Edit your spec file to include location of missing module, for example for pyqt5

```
...
datas=[('c:/users/username/projectvenv/Lib/site-packages/PyQt5/Qt/bin/*','PyQt5/Qt/bin
↪')],
...
```

- then compile your project via .spec file `pyinstaller project.spec`. If you dont have a spec file, create one via `pyi-makespec project --onefile`

2.2.17 lib - PyQt5 (GUI)

Qt framework is a very powerful cross-platform GUI builder. It is written in C++ and it was ported, over to python as pyqt. The python documentation is nearly non-existent since it would sort of be a duplicate of the Qt C++ docs (PyQt4 if PyQt5 data is missing, PyQt5 and Qt). This package has a very steep learning curve, so take it slow and try to get used to reading the C++ docs.

Installation

Note if you get a `Could not find a version that satisfies the requirements` during a pip install, then your current python version is not supported by pyqt.

```
pip install pyqt5 pyqt5-tools
```

Designer (GUI builder)

PyQt comes with an awesome drag and drop GUI builder `designer.exe`. The tool (and all other pyqt .exe) will be placed in:

- lib/site-package/pyqt5_tools/Qt/bin
- or if you are using virtualenv, it will simply be under Scripts

1) Construct your GUI by drag/drop method and save it as a `.ui`
2) Convert your `.ui` to python code with `pyuic5.exe`

```
# -x to make it executable (creates __name__ == "__main__")
# -o to specify output filename
pyuic5.exe -x -o outputfilename.py designerfilename.ui
```

Designer to Python code setup

An efficient way to use Qt Designer is export out the widget code then without alterations of your designer created code we import it into the logic modules. This will save a lot of time when we have to go back to designer and adjust something or add a new widget. We simply re-export out the python code and the import takes care of the rest. The following example is a good starter code that handles the exported `form.py` designer exported python code.

- QtWidgets: App > MainWindow > all widgets QtWidgets
- QtCore: brains of qt QtCore
  - Qt: misc qt library items(ex: keys, mouse) Qt
    - Keys: QKey
    - Mouse Keys: QMouseButton
  - QPoint: hold point properties (ex: position) Qpoint
  - QEvent: all event types, but not sensor (ex: KeyPress) QEvent
- QtGui: event sensors and graphical editor (ex: colors, fonts etc) QGui

```
# form.py is the designer exported python code
from form import Ui_MainWindow
# QtWidgets is the collection of all Qt windows (QApplication > QMainWindow > widgets,
↪ events)
# QtCore is the collection of keyboard/mouse/event types
# QtGui is the collection of event sensors and graphical editors like color/font
from PyQt5 import QtWidgets, QtCore, QtGui
# sys is call to handle any arguments passed in from the terminal (optional)
import sys

class Ui(QtWidgets.QMainWindow, Ui_MainWindow):
    def __init__(self, *args, **kwargs):
        # initializes QMainWindow object (so that we can call: MainWindow.attribute)
        QtWidgets.QMainWindow.__init__(self, *args, **kwargs)
        # sets up the Designer widgets that we imported
        self.setupUi(self)

        # to enable event handling (if this not stated python will garbage collect␣
↪all events)
        self.installEventFilter(self)

    # overwrite the Qt event filtering method
    #   (by default it is empty so we edit it to handle key presses
    def eventFilter(self, source, event):
        # lets see how we setup a custom key event
        if (event.type() == QtCore.QEvent.KeyPress and
                event.key() == QtCore.Qt.Key_A):
            print('you presses the "A" key')

        # first check if a key was pressed, then check if that event matches ctrl+c
        #  which is already built into qt as QKeySequence.Copy
        if (event.type() == QtCore.QEvent.KeyPress and
                event.matches(QtGui.QKeySequence.Copy)):
            # now pipe the event to any method to logic handling
            self.customcopy()

        # this is to overwrite the existing event filter method
```

(continues on next page)

(continued from previous page)

```
        return super(Ui, self).eventFilter(source, event)

    # our custom method to handle what happens when we hit ctrl+c
    def customcopy(self):
        print("you hit ctrl+c")

if __name__ == "__main__":
    # create an instance of Qt (pass in sys.argv allows args to be passed it from␣
↪terminal)
    app = QtWidgets.QApplication(sys.argv)
    # initialize the MainWindow
    gui = Ui()
    # shown the MainWindow
    gui.show()
    # app.exce_() runs the mainloop, and returns 0 for no error, 1 for error
    sys.exit(app.exec_())
```

Events

- paintEvent
- resizeEvent
- keyPressEvent and keyReleaseEvent
- contextMenuEvent
- mouseMoveEvent and mouseReleaseEvent and mouseDoubleClickEvent

Using Builtin Signals

Qt widgets already come with a ton of handy signals already coded up that handle events for you. See the Custom Signal/Connect/Emit Setup section to get a in depth walkthrough on how a signal works but in short, a signal is already hocked up event handler for a widget action (like the press of a button). You only have to connect up what happens when a specific signal is emitted (an event happens like pressing a button) and the rest is taken care of for you (for builtin signals). Lets see how to hock up a builtin signal from `QLineEdit` text filed to a `QLabel` text when the "Enter" is pressed from the `QLineEdit` widget:

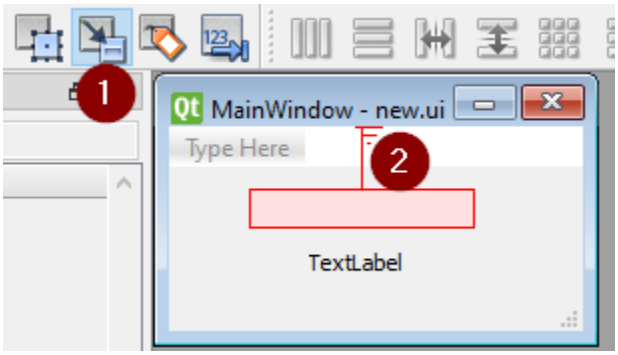
```
from PyQt5 import QtCore, QtGui, QtWidgets


class Ui_MainWindow(object):
    def setupUi(self, MainWindow):
        MainWindow.setObjectName("MainWindow")
        MainWindow.resize(207, 102)
        self.centralwidget = QtWidgets.QWidget(MainWindow)
        self.centralwidget.setObjectName("centralwidget")
        self.lineEdit = QtWidgets.QLineEdit(self.centralwidget)
        self.lineEdit.setGeometry(QtCore.QRect(40, 10, 113, 20))
        self.lineEdit.setObjectName("lineEdit")
        self.label = QtWidgets.QLabel(self.centralwidget)
        self.label.setGeometry(QtCore.QRect(70, 40, 47, 13))
        self.label.setObjectName("label")
        MainWindow.setCentralWidget(self.centralwidget)
        self.menubar = QtWidgets.QMenuBar(MainWindow)
```

(continues on next page)

(continued from previous page)

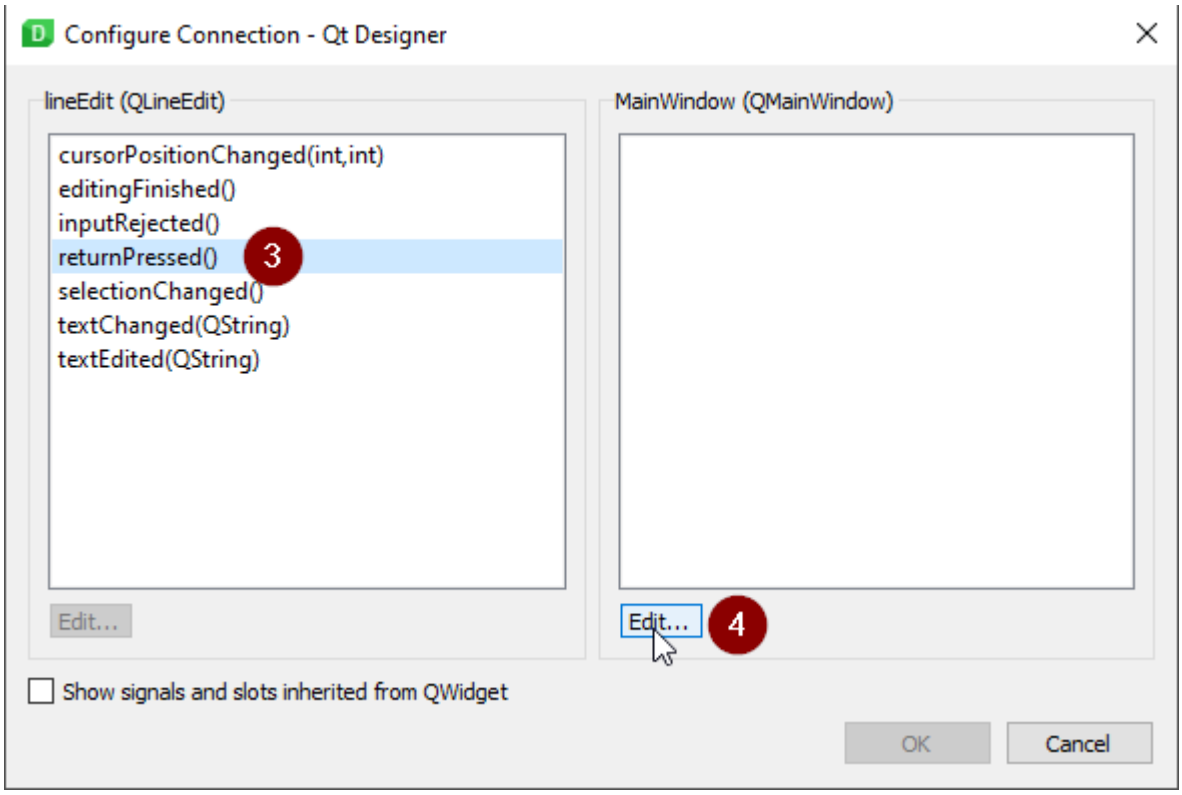
```
        self.menubar.setGeometry(QtCore.QRect(0, 0, 207, 21))
        self.menubar.setObjectName("menubar")
        MainWindow.setMenuBar(self.menubar)
        self.statusbar = QtWidgets.QStatusBar(MainWindow)
        self.statusbar.setObjectName("statusbar")
        MainWindow.setStatusBar(self.statusbar)

        self.retranslateUi(MainWindow)
        QtCore.QMetaObject.connectSlotsByName(MainWindow)

        # setup the connection from our QLineEdit widget to our method
        self.lineEdit.returnPressed.connect(self.CustomMethod)

    def retranslateUi(self, MainWindow):
        _translate = QtCore.QCoreApplication.translate
        MainWindow.setWindowTitle(_translate("MainWindow", "MainWindow"))
        self.label.setText(_translate("MainWindow", "TextLabel"))

    # our custom method
    def CustomMethod(self):
        # grab the text from the text field
        text = self.lineEdit.text()
        self.label.setText(text)


if __name__ == "__main__":
    import sys
    app = QtWidgets.QApplication(sys.argv)
    MainWindow = QtWidgets.QMainWindow()
    ui = Ui_MainWindow()
    ui.setupUi(MainWindow)
    MainWindow.show()
    sys.exit(app.exec_())
```

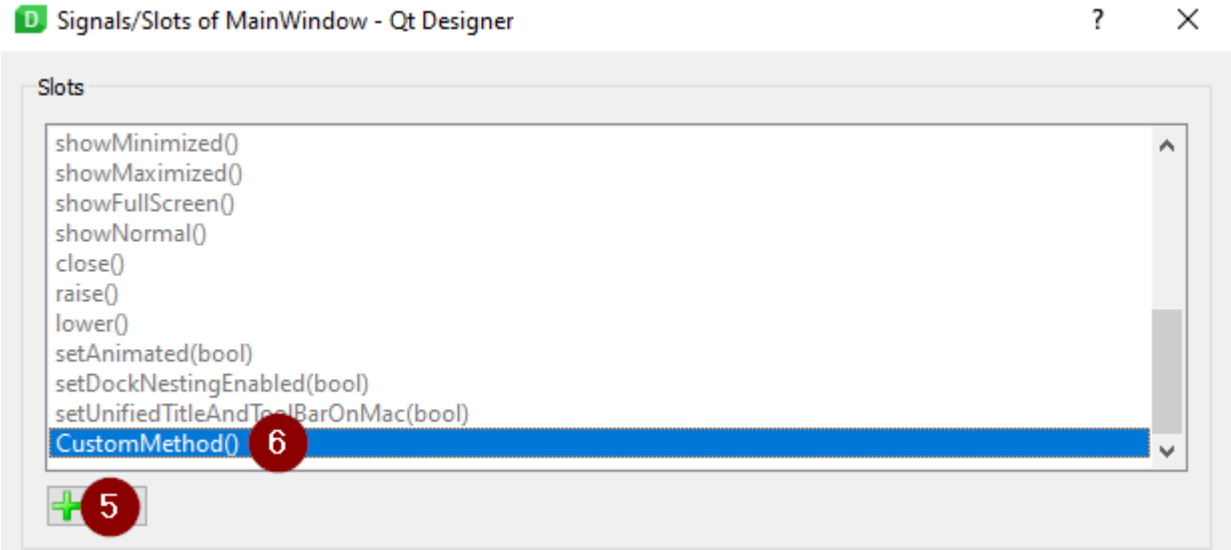
We can also have Qt Design create the `self.lineEdit.returnPressed.connect()` line for us but it doesn't really save much time. Note that we will have to replace the method name since pyqt Designer does not allow us to type "self.Here are the steps:



Custom Signal/Connect/Emit Setup

Signals are a great way to jump in and out of function when a certain event or condition was satisfied. As with any problem, this what a signal does can also be achieved without ever using signals but signals can make more of a logical sense. There are 4 pieces to a signal setup/use:


Configure Connection - Qt Designer
lineEdit (QLineEdit)
cursorPositionChanged(int,int)
editingFinished()
inputRejected()
returnPressed()
selectionChanged()
textChanged(QString)
textEdited(QString)
3
Edit...
MainWindow (QMainWindow)
Edit...
4
Show signals and slots inherited from QWidget
OK
Cancel


Signals/Slots of MainWindow - Qt Designer
Slots
showMinimized()
showMaximized()
showFullScreen()
showNormal()
close()
raise()
lower()
setAnimated(bool)
setDockNestingEnabled(bool)
setUnifiedTitleAndToolBarOnMac(bool)
CustomMethod()
6
5

1) Signal: Class Attribute; Defines the signal name, and sets up argument types (types must be setup)
2) Define Slot: Class Method; Defines the method that is called when a signal is emitted
3) Connect: Inside __init__; Connects the Signal Class Attribute to the Class Method
4) Emit: A Call; Emit a signal

```
# form.py is the designer exported python code
from form import Ui_MainWindow
from PyQt5 import QtWidgets, QtCore, QtGui
import sys

class Ui(QtWidgets.QMainWindow, Ui_MainWindow):
    # STEP 1: Define a "SIGNAL", and define the type or argument that is being passed
    #  in this example: we can pass a bool and str argument when a emit occurs
    a_key_pressed = QtCore.pyqtSignal(bool,str)


    def __init__(self, *args, **kwargs):
        QtWidgets.QMainWindow.__init__(self, *args, **kwargs)
        self.setupUi(self)
        self.installEventFilter(self)

        # STEP 3: "CONNECT" a signal to a "SLOT"
        a_key_pressed.connect(self.slot_a_key_pressed)


    # overwrite the Qt event filtering method
    #   (by default it is empty so we edit it to handle key presses
    def eventFilter(self, source, event):
        # lets see how we setup a custom key event
        if (event.type() == QtCore.QEvent.KeyPress and
                event.key() == QtCore.Qt.Key_A):
            print('you presses the "A" key')
            # STEP 4: "EMIT" the signal in practice
            a_key_pressed.emit(True,"emitted signal A")

    # STEP 2: define a "SLOT" that handle what happens when the signal is emitted
    def slot_a_key_pressed(self,arg1,arg2):
        print(f"It is {arg1} that we {arg2}")


if __name__ == "__main__":
    # create an instance of Qt (pass in sys.argv allows args to be passed it from␣
↪terminal)
    app = QtWidgets.QApplication(sys.argv)
    # initialize the MainWindow
    gui = Ui()
    # shown the MainWindow
    gui.show()
    # app.exce_() runs the mainloop, and returns 0 for no error, 1 for error
    sys.exit(app.exec_())
```

Path File Browser

There are several file/folder browser dialogs available to the user:

- `QFileDialog.getExistingDirectory(self, title, path, filter)` where filter: "Images (*.png *.jpg);; Text(.txt)"

```
# NOTE: this is another method to the example shown above under "Designer to Python␣
↪code setup"

def getpath(self):
    path = QtWidgets.QFileDialog.getExistingDirectory(self, 'Select Directory')
    return path
```

- `QFileDialog.getOpenFileName(self, title, path, filter)` similar to directory expect this opens the file for streaming this can return a returns a list no matter what, if a single file was selected or multiple.

```
# NOTE: this is another method to the example shown above under "Designer to Python␣
↪code setup"
def openfile(self):
    filename = QtWidgets.QFileDialog.getOpenFileName(self, 'Select File')
    if filename[0]:
        with open(filename[0], 'r') as f:
            data = f.read()
```

- `QFileDialog.getSaveFileName(self, title, path, filter)` similar to file expect this opens the file for streaming

```
# NOTE: this is another method to the example shown above under "Designer to Python␣
↪code setup"
def openfile(self):
    filename = QtWidgets.QFileDialog.getSaveFileName(self, 'Select File to Save')
    if filename[0]:
        with open(filename[0], 'w') as f:
            f.write(data)
```

MessageBox Popup

```
# NOTE: this is another method to the example shown above under "Designer to Python␣
↪code setup"

# the following is useful as error handling popup
try:
    # some code
except Exception as e:
    msgbox = QtWidgets.QErrorMessage(self)
    msgbox.showMessage(str(e))
```

Common Widgets And A Short Description

QLabel

A un-editable text field has the following methods

- `setText()` assign text to the Label
- `clear()` clear the text from the Label

QLineEdit

An editable text field has the following methods

- `setEchoMode(int)` possible inputs:
  - 0: Normal, what you type is what you see
  - 1: NoEcho, you cannot see what you type but the text is still stored
  - 2: Password, each character types is instead replaced by "*"
  - 3: PasswordEchoOnEdit,it displays the characters while typing but is then "*" out afterwards
- `maxLength()` specify how many characters can be typed into the text field
- `setText()` set a default text
- `text()` get the text out of the text field
- `clear()` clears text field
- `setReadOnly()` text field cannot be edited but it can be copied
- `setEnabled()` by default = `True` but can be passed a `False` to disable from edit/copy
- Signals:
  - `QLineEdit.textChanged.connect(custom_method_pipe)` when text is changed
  - `QLineEdit.returnPressed.connect(custom_method_pipe)` when enter is pressed from textbox

QPushButton

Simple on/off button that emits a signal when clicked, with the following

QTableWidget

- `setRowCount()` redefine how many rows there are in the table (similar for column)
- `rowCount()` returns the number of rows in the table (similar for column) note this is not the row that contain data, but all rows
- `clear()` clears all content from the entire table
- `setItem(row,col,QtWidgets.QTableWidgeItem(data))` where row and column are `int` and `data` is a `str`

```
# NOTE: this is another method to the example shown above under "Designer to Python␣
↪code setup"

# set cell value
def mycellsetter(self, value):
    # input value must be a string
    row = 0
    col = 0
    self.table.setItem(row,col,QtWidgets.QTableWidgetItem(str(value)))

# to get cell value
```

(continues on next page)

(continued from previous page)

```
def mycellgetter(self):
    row = 0
    col = 0
    # return values will always be strings
    return self.table.item(row, col).text()

# to iterate through a tableWidget
def tableiter(self):
    maxcol = self.table.model().columnCount()
    maxrow = self.table.model().rowCount()
    for c in range(maxcol):
        for r in range(maxrow):
            # note that empty cells show up as None type
            if self.table.item(r,c) != None:
                # to get the actual value stored we have to call .text() on the␣
↪current cell
                self.table.item(r,c).text()

# to copy from table
def copySelection(self):
    # note this is tablename specific (table name = "table")
    selection = self.table.selectedIndexes()
    if selection:
        rows = sorted(index.row()) for index in selection)
        columns = sorted(index.column() for index in selection)
        rowcount = rows[-1] - row[0] + 1
        colcount = columns[-1] - columns[0] + 1
        table = [[''] * colcount for _ in range(rowcount)]
        for index in selection:
            row = index.row() - rows[0]
            column = index.coumn() - columns[0]
            table[row][column] = index.data()
        stream = io.StringIO()
        csv.writer(stream, delimiter='\t').writerows(table)
        QtWidgets.qApp.clipboard().setText(stream.getvalue())

# to paste to table
def pasteSelection(self):
    # note this is table name specific (table widget name = "table")
    selection = self.table.selectedIndexes()
    model = self.table.model()

    if selection:
        buffer = QtWidgets.qApp.clipboard().text()
        rows = sorted(index.row() for index in selection)
        columns = sorted(index.column() for index in selection)
        reader = csv.reader(io.StringIO(buffer), delimiter='\t')
        if len(rows) == 1 and len(columns) == 1:
            for i, line in enumerate(reader):
                for j, cell in enumerate(line):
                    model.setData(model.index(row[0] + 1, columns[0] + j), cell)
        else:
            arr = [[cell for cell in row] for row in reader]
            for index in selection:
                row = index.row() - rows[0]
                column = index.column() - columns[0]
                model.setData(model.index(index.row(), index.column()),␣
↪arr[row][column])
```

(continues on next page)

(continued from previous page)

Indexing A QTabWdiget

```
def tabpiper(self):
    if self.yourtabwidgetname.currentIndex() == 0:
        print('you are on the first tab')
    elif self.yourtabwidgetname.currentIndex() == 1:
        print('you are on the second tab')
```

PyInstaller Packing TroubleShooting

Dealing with "ImportError: unable to find QtCore.dll on PATH"

- Run on pyinstaller 3.5 and PyQt5 5.12.3 (PyInstaller Link)
- Create spec file via (pyi-makespec filename.py)
- Add to gui.spec datas=[('fullpath/site-packages/PyQt5/Qt/bin/*','PyQt5/Qt/bin')] then run pyinstaller gui.spec –onefile

GUI Lockup - Multithreading

Execute multiple tasks without locking up the GUI. Threading has a few parts:

- class initialization and class instance: where we feed information to the thread class, like the GUI window
- class run method and class start(): `threadname.start()` calls the `run` method from the thread class
- threads join: join all of the threads together

```
# Qt Designer Output of 2 progress bars
from PyQt5 import QtCore, QtGui, QtWidgets


class Ui_Dialog(object):
    def setupUi(self, Dialog):
        Dialog.setObjectName("Dialog")
        Dialog.resize(400, 300)
        self.progressBar = QtWidgets.QProgressBar(Dialog)
        self.progressBar.setGeometry(QtCore.QRect(130, 80, 118, 23))
        self.progressBar.setProperty("value", 0)
        self.progressBar.setObjectName("progressBar")
        self.progressBar_2 = QtWidgets.QProgressBar(Dialog)
        self.progressBar_2.setGeometry(QtCore.QRect(140, 160, 118, 23))
        self.progressBar_2.setProperty("value", 0)
        self.progressBar_2.setObjectName("progressBar_2")

        self.retranslateUi(Dialog)
        QtCore.QMetaObject.connectSlotsByName(Dialog)

    def retranslateUi(self, Dialog):
        _translate = QtCore.QCoreApplication.translate
        Dialog.setWindowTitle(_translate("Dialog", "Dialog"))
```

(continues on next page)

(continued from previous page)

```
# our code for threading on 2 progress bars without locking up the GUI
import sys
import threading
import time
from PyQt5.QtWidgets import QDialog, QApplication

class GUI(QDialog):
    def __init__(self):
        super().__init__()
        self.ui = Ui_Dialog()
        self.ui.setupUi(self)
        self.show()
class aThread (threading.Thread):
    counter=0
    def __init__(self, gui, ProgressBar, steps):
        threading.Thread.__init__(self)
        self.gui=gui
        self.counter=0
        self.steps = steps
        self.progreassBar=ProgressBar
    def run(self):
        print ("Starting " + self.name)
        while self.counter <=100:
            time.sleep(0.5)
            self.progreassBar.setValue(self.counter)
            self.counter+=self.steps
        print ("Exiting " + self.name)

if __name__=="__main__":
    app = QApplication(sys.argv)
    gui = GUI()
    thread1 = aThread(gui, gui.ui.progressBar)
    thread2 = aThread(gui, gui.ui.progressBar_2)
    thread1.start() # to start the thread (calls .run())
    thread2.start() # to start thread2 (calls .run())
    gui.exec()  # this is to keep the gui window responsive
    thread1.join()  # bring back the tread and merge data
    thread2.join()
    sys.exit(app.exec_())
```

2.2.18 lib - pytest

The concept of writing code to test code can be a bit bizarre at first, but the idea is to test the behavior of a function or integration of functions with a set of inputs that have a known desired output value (ie. add 1 + 3; the inputs are 1 and 3, and we know that the correct answer should be 4).

Why write tests:

- Code reliability. Does the code work the way it is was intended?
- Code maintainability. During refactoring or feature addition/removal does the code still do what it was supposed to do?
- Code longevity. During version updates, outputs of dependent packages can change. A properly tested code will ensure that all outputs of dependent packages still work for your code.

- Tested code will yield cleaner code as well. It will be come very apparent in practice that a single function that does everything will be much hard to test (and read) than that same code refactored into many smaller functions with explicit output.

Assertions: Python.org

Setup your first test

Although it is not necessary to create tests in another file, it is highly encouraged to keep the code base nice and clean.

- The code we want to test: let's say the the following is in a file `mycode.py`

```
# filename: mycode.py

def adder(x,y):
    return x + y
```

- The test code: let's say the following is in a file `test_mycode.py` It is a good idea to keep write a "test" file for each "code" file. Code files should also be short (not 1000s of lines).

```
# filename: test_mycode.py

# import the builtin test library
import unittest
# import your code module (mycode.py, assuming both mycode.py and test_myode.py is in␣
↪the same folder)
import mycode


# setup the test class
class test_mycode(unittest.TestCase):

    # setUp is optional: but if you are re-using inputs this saves you from retying␣
↪them
    def setUp(self):
        self.input1 = 10
        self.input2 = 20

    # pytest looks for all methods that start with "test_"
    def test_one(self):
        # there are many different asserts, see full list above
        #  here we are testing if our function "adder" adds 10+20 correctly and␣
↪equals 30
        self.assertEqual(mycode.adder(10,20),30)

    # we can write as many tests as we like,
    #   here is the same test input/out but with our setUp variables
    def test_two(self):
        self.assertEqual(mycode.adder(self.input1,self.input2),30)
```

- To run pytest, we type the following in the terminal

```
python -m pip install pytest pytest-cov
python -m pytest --cov-report=html --cov='.'
```

- The output will look something like:

```
============================= test session starts ==============================
platform win32 -- Python 3.8.0, pytest-5.2.2, py-1.8.0, pluggy-0.13.0
rootdir: C:\Users\yourusername\Desktop
collected 2 items

test_mycode.py ..                                                        [100%]

============================== 2 passed in 0.03s ===============================
```

How to check if a function raises an error

Reusing the same example from `mycode.py`

```
# filename: test_mycode.py

import unittest
import mycode


class test_mycode(unittest.TestCase):

    def test_error(self):
        # to test a error raise, we have to enclose the code being testing a "with"␣
↪block
        #  here we are testing if our code raises a TypeError when adding 10 + "20"␣
↪as it should
        with self.assertRaises(TypeError):
            mycode.adder(10,"20")
```

How to report out code test coverage

Code test coverage writes out a detailed report on what percent of your code the test actually executed.

```
python -m pytest --cov-report=html --cov='.'
```

You can also write out a single `xml` coverage file. This is useful for CI (continuous integration) since you only have to point your upload/file to 1 file.

```
python -m pytest --cov-report=xml --cov='.'
```

To mock file-read without an actual file

```
f = io.StringIO("text\n")

f.readline()
>>> "text"
```

2.2.19 lib - Sphinx (code documentation)

1) Create a virtualenv, see *lib - virtualenv*
2) Active virtualenv and add sphinx `pip install sphinx`

3) Change dir to your 00_project folder

4) Run `sphinx-quickstart` from the terminal (you will want to separate source and build for a cleaner structure)

5) To install a custom sphinx theme: `pip install sphinx_rtd_theme`

6) Edit theme on your `source/conf.py` file to be `sphinx_rtd_theme`

7) Create your docs

    7.1) Make sure your files have headers (line 1: header text, line2: =======)

    7.2) Make sure you add each filename to your index.rst otherwise it is not going to show up as an index

8) To create your html files `./make.bat html`

9) To clean up your builds before git push `./make.bat clean`

10) Push your docs to your git (make sure you do a `pip freeze > requirements.txt` because readthedocs might fail for using the incorrect version)

11) Setup your commit hock with readthedocs (login to readthedocs, add your github project, then build). After your build is complete, each git push will auto-trigger a webhock to readthedocs

Text Manipulations

Ref Sphinx rst docs: - Link1 - Link2

- To italicize text: `*text*`
- To bold text: `**text**`
- Subscript/Superscript: `:sub:`yourtext`` or `:sup:`yourtext``
- To in-text code highlight: ``` ``text`` ```
- Important messages: `.. note:: .. warning:: .. deprecated:: .. seealso::`
- Internal Links: `:doc:`filename``
- External Links: `` `linktext <https://google.com>`_ ``
- Section Links within the same doc
    1) put a `.. _ref1:` above the header you want to ref (make sure there is an empty line between the header and _ref1)
    2) call the link by: `:ref:`ref1``
- Citations: to define it `.. [refname1] Title` then to use it in text: `[refname1]_`
- Today's date in text `|today|`
- Bullets, there are a few not so intuitive conventions about bullets, see below:

```
- this is a bullet

- this is another bullet

- this is a bullet
on multi lines

1) this is a numbered bullet
```

(continues on next page)

(continued from previous page)

```
    1.1) this is a nested bullet
    with multi lines

    1.2) here is another nested bullet
```

Header Types

- Title: ###
- Section: ===
- SubSection: —
- 2SubSection: ^^^
- 3SubSection: “””

Sphinx toctree

1) The `.. toctree::` must be present in your top level index.rst file. The links will show up on both your page and on the left quickbar
2) The sub-folder trees also need to have a index.rst with `.. toctree::` to properly get nested tree reference links, see example below:

```
on the top-level index.rst
.. toctree::

    python/index
    VBA/index
    Shell/index
```

```
in the subdirectory /python/ we have a index.rst, it needs to have a header and a sub-
↪toctree

Python Guide
============

.. toctree::

    virtualenv
    sphinx
```

Code-Blocks

Note: “.. code-block::” has to have an empty line above and below it, AND empty line after your code. The code has to be on the same indent level as “:linenos”

```
.. code-block:: shell
    :linenos:
    :lineno-start: 10
    :emphasize-lines: 3,5

    some shell code
```

Code-Auto-Doc

1) Uncomment the following from your config file:

```
import os
import sys
sys.path.insert(0, os.path.abspath('.'))
```

2) In your desired .rst file, add the following (where each function/class is the member):

```
.. automodule::
    :members: foo, bar

.. autoclass::
    :members:

.. autofunction:: folder.file.functionname
```

Figures

```
.. figure:: pic.png
    :scale: 50%
    :alt: Alternative text if image does not load, spoken by application for visually␣
↪impaired
    :align: center

    This is caption text
```

2.2.20 lib - Subprocess

To run another python script within another python script

```
import subprocess

subprocess.run("python script.py")
```

2.2.21 lib - virtualenv

Developers use virtual environments for 2 main reasons:

- Isolating different projects on 1 machine and managing their dependencies
- Supporting various versions of python and/or other 3rd party libraries for a single project

There are a few options out there for virtual environments:

- venv: comes with python and it is very basic
- virtualenv: pip installed library, i feel like i had better success using pyinstaller with virtualenv vs venv
- pipenv: pip installed library, gives better dependency descriptions and git-branch description
- conda: shipped with anadonda, very similar to virtualenv

###

1) pip install virtualenv (version 16.1.0 for win10 build compatibility) in your global site-packages

```
pip install virtualenv==16.1.0
```

2) Create your git repo online and clone it down (see *tool - Git*)

2) Browse into your git repo and create your virtualenv from terminal:

```
python -m virtualenv venv38
```

Note as a convention, I recommend placing the venv inside your git repo folder so that everything is together. This setup integrates really nicely with Editors like PyCharm where the project recognizes that you created a virtualenv inside your git repo folder.

This will create you a folder with bear minimum python packages (this is nice if you would like to pyinstaller package up your work that ONLY use the packages required for your project). The folder created has a bunch of folders.

- *Lib/Site-Package*: has your packages installed, if you want to manually add a module - this is where you would place it
- *Scripts*: has your activate/deactive, and all your run files like python.exe, pip etc.

3) Activate your virtualenv (with Gitbash, assuming you are one dir higher than your virtualenv)

```
source venv_projectname/Scripts/activate
```

4) Deactive

```
deactivate
```

5) To ensure we are not pushing unnecessary items to our git repo, we can create a `.gitignore` file. Create a `.gitignore` (from terminal: `touch .gitignore` and you will most likely want the following in it:

```
Include
Lib
Scripts
tcl

__pycache__
*.pyc

build
dist
venv_projectname
```

2.2.22 tool - Git

- Git is what's called a Version Control System (VCS).

- Git is not the same as GitHub or GitLab. Git is a tool in itself. It is a tool that takes a complete snapshots of your work with each commit ("save") thereby allowing the user to roll an entire project forward and aft in time without having to worry about piecing the project back together.
- Git can be downloaded for windows in a form of a linux emulator (GitBash), whilst mac and linux distributions may already have it installed. The user can work with it 100% offline if that is desired, however cloud providers such as GitHub and GitLab offer a collaborative online experience for a team of developers.

Setup

- Set up your username in the terminal `git config --global user.name "Viktor Kis"`
- Set your email address: `git config --global user.email name@example.com`
- Set your editor: `git config --global core.editor vim`
- Check your user setttings: `git config --list`

Step-by-Step GitHub Repo

1) Create a free github account at: github.com
2) On the top right click your `user icon` and a drop-down should appear > Click `Your repositories`
3) On your `Repositories` tab towards the right, there should a green `New` icon, click it.
4) Give your new repo a name and hit `Create repository`
5) Go back to the `Repositories` tab and select your new repo. On your repo page, towards the right there there should be a green `Clone or download` button, click it and copy the http link.
6) On your local machine, open up a terminal and type `git clone http-for-your-repo`
7) You now have a successful link up to your online repo from your local. Let's step through a basic upload

    7.1) Add a new file to your repo folder (inside your project folder that was cloned down)

    7.2) Add it to queue: `git add .`

    7.3) Add a commit message: `git commit -m "initial demo upload"`

    7.4) Push it to the github: `git push`

Commands

General

- To initialize a folder: `git init` or clone an existing repo `git clone url`
  - Note that `git clone url` sets up your remote link, `git init` does not (see remote below to link up a initialized project)
- To add file(s) in queue for save: all files `git add .` single file `git add filename`
- To remove an already added file from queue: `git reset`
- To commit a change: `git commit -m "msg with your commit"`
- To push a commit to the cloud: `git push`
- To pull the latest data from the branch: `git pull` or explicitly `git pull origin master` (note that `git fetch` works similarly, however it does not merge the work with your local changes)

- To completely overwrite local files with server files: `git reset --hard origin/master`

Status

- To check change status: `git status`
- To check the past commit logs: `git log --graph` to limit the log `git log --since=2.weeks`

Branches

- To create a new branch: `git checkout -b branchname`
- To switch between branches: `git checkout branchname`
- To merge a branch onto another: `git merge`

Ignore Files

- To create an ignore files/file-types, create a .gitignore: `touch .gitignore`
- By practice developers should only commit source files (no binaries, no .pyc files, no config files and etc.) ex; `*.pyc`

Remote

- To add a remote link `git remote add user_defined_remote_name url`

```
git clone github.com/project.git
git remote -v
>>> origin https://github.com/project.git (fetch)
>>> origin https://github.com/project.git (push)
# to add a link to someone's fork of the project for example, we would:
git remote add fork_vik https://github.com/vik/project.git
git remote -v
>>> origin https://github.com/project.git (fetch)
>>> origin https://github.com/project.git (push)
>>> fork_vik https://github.com/vik/project.git (fetch)
>>> fork_vik https://github.com/vik/project.git (push)
# now we have to option to pull/fetch from a fork onto our local project
git pull fork_vik master
```

Branch

- To create a branch: `git branch branch_name`
- To work/change your current branch: `git checkout branch_name`

Common Issues

- "failed to push some refs to repo, tip of your current branch is behind"
  - Cause: there were changes to the remote repo that you dont have (this could be file or history log change)

- Fix: run a `git pull` and resolve the conflicts

2.2.23 tool - pip (Package Installer for Python)

General

- To show list of packages available: `pip list`
- To check list of packages available within python runtime: `help('modules')`
- To check the version of a package within python: first import the package `import pandas` then check version `pandas.__version__`
- To upgrade a package: `pip install packagename -U`
- To force install a specific version of a package: `pip install "pandas=0.19.3" --force-reinstall`

Packaging

```
# save a text file of all required packages and their versions
pip freeze > requirements.txt

# to install these packages on a server or another computer
pip install -r requirements.txt
```

TroubleShooting - Nothing happens when i type `pip` on terminal

1) Double check that you have your environment path setup correctly. It should be pointing to your "Scripts" folder, see :doc:`10min_python_starters_kit`
2) If simply `pip` doesnt work on your terminal, then typing out the command through python is a good backup. `python -m pip`

2.2.24 tool - PyCharm (IDE)

General

- To comment out lines: `ctrl + /`
- To auto PEP8 (code cleanup/format): `ctrl + alt + L`
- To enable Crtl+Mouse Wheel text zoom: File > Settings > Editor > General > Mouse > Change Font size...
- To change max character line: File > Settings > Editor > Code Style > Right Margin (columns) 120 default
- To search PyCharm commands: `ctrl + shift + a`
- To close file tabs: `ctrl + F4`
- Expand/Contract Code Blocks for a single code block: `ctrl + +` and `ctrl + -` (courser within code block)
- Expand/Contract Code Blocks for a all code block: `ctrl + shift + +` and `ctrl + shift + -`

New Project Setup

1) If project folder already exists: File > Open > Navigate to folder
2) Configure your python environment: File > Setting > Project > Project Interpreter > Gear on top right > Add >> Virtualenv Environment > 2 options here...
    1) Create a virtualenv (highly recommended, for pip version control) Either link to an existing virtualenv python.exe (this will be under ProjectNameVenv/Scripts/) or create a virtualenv directly within PyCharm (note that you have to open to inherit all global packages, make sure this is unchecked before creating the virtualenv)
    2) Link to global python interpreter (access to all global pip)

Terminal

- To configure terminal start directory:

File > Settings > Tools > Terminal > Start directory

- To configure a custom terminal emulator (like gitbash) inside pycharm terminal tab:

File > Settings > Tools > Terminal > Shell path: `"C:\Program Files\Git\bin\sh.exe" --login`

2.2.25 tool - VScode (IDE)

Light weight, simple IDE built by microsoft.

- increase/decrease font size: `crtl` + = and `ctrl` + -
- run file in debug: `F5`
- set the default terminal (to git bash): `ctrl` + `shift` + `P` then select default terminal
- redefined keybindings (settings - the gear in bottom left corner - keyboard shortcuts):
    - copy line down: `ctrl + d`
    - delete line: `ctrl + y`
    - fold (collapse code): `ctrl + -`
    - fold-all (collapse code): `ctrl + shift + -`
    - unfold (collapse code): `ctrl + =`
    - unfold-all (collapse code): `ctrl + shift + =`

2.2.26 tool - vim (text editor)

Why use vim? - vim is installed on every linux machine, therefore knowing how to work it will ensure

> that you are able to work any where. New machine, coworker's machine, remote servers (where your favourite editor cannot be installed/used)

- How to start vim on terminal: `vim` or `vim filename`
- **By default vim opens in `command mode` you cannot edit the file from this mode** to enter `insert mode` press `i` or `insert` key. To get back to `commad mode` press `esc`
- command window - write to file and quit: `:wq`

- command window - quit without saving: `q!:`
- command window - increase font size: `ctrl` + `=`
- command window - decrease font size: `ctrl` + `-`
- command window - add syntax highlighting `syntax on` `syntax off`
- command window - add line numbers `set number` `set nonumber`
- command window - jump to the beginning of a file `gg` jump to the end of a file `shift` + `g`
- command window - jump to a line number `:100` to jump to line 100
- command window - undo last command `u`
- insert mode - increase indent: `ctrl` + `t`
- insert mode - decrease indent: `ctrl` + `d`

In addition you can launch a vim tutorial in your terminal by typing `vimtutor` in your terminal

searching

1) enter `command window`
2) type `/` followed by the character or string to search, ex: `/hello world` then press `enter`
3) press `n` to search forward or `N` to search backward

2.2.27 DevOps - Continuous Integration (CI)

Continuous Integration (CI) is used for various Development Operations (DevOps), such as running unittests, integration tests, generating code monitor reports like coverage, and lint for PEP8 code quality, and much more. In this section, we will focus on automating our testing with CI on `circleci` and utilizing code coverage report hosting on `codecov`. There are many options out there for CI hosts, but I found these to be the easiest to use.

0.) Navigate to circleci and codecov and connect your github account with these APIs.

1.) Create a `.circleci` folder in your top level repo directory

2.) Create a `config.yml` file within the `.circleci` folder and paste the following in it:

```
# definition of circleCI version
version: 2.1
orbs:
  # defines the connection to codecov's API
  codecov: codecov/codecov@1.1.1


# define workflow of executions
workflows:
  version: 2.1
  # name your workflow
  test:
    # define the job names under test
    jobs:
      - test27
      - test38

jobs:
```

(continues on next page)

(continued from previous page)

```
test27:
  docker:
  # create a python image on the server
    - image: circleci/python:2.7

  # change server dir to repo where we clone down the repo
  working_directory: ~/repo

  steps:
    # download repo
    - checkout

    # install dependencies
    - run:
        name: install dependencies
        # note on linux virtualenv is in venv/bin/activate not venv/Scripts/activate
        command: |
          python -m pip install virtualenv
          python -m virtualenv venv
          . venv/bin/activate
          python -m pip install pytest pytest-cov

    # run tests
    - run:
        name: run tests
        # note you have to reactive on each -run
        command: |
          . venv/bin/activate
          python -m pytest

test38:
  docker:
    - image: circleci/python:3.8

  # change server dir to repo where we clone down the repo
  working_directory: ~/repo

  steps:
    # download repo
    - checkout

    # install dependencies
    - run:
        name: install dependencies
        # note on linux virtualenv is in venv/bin/activate not venv/Scripts/activate
        command: |
          python -m pip install virtualenv
          python -m virtualenv venv
          . venv/bin/activate
          python -m pip install pytest pytest-cov

    # run tests
    - run:
        name: run tests
        # note you have to reactive on each -run
        command: |
          . venv/bin/activate
```

(continues on next page)

(continued from previous page)

```
            python -m pytest --cov-report=xml --cov='.'
            ls

    # point codecov to your coverage file and upload it to their API
      - codecov/upload:
          file: coverage.xml
```

2.3 VBA Guide

2.3.1 10min - VBA Starter Kit

This starter kit is meant for readers that have never used VBA and have very little knowledge of programming concepts.

Initial Set-Up

All Microsoft Office Applications can be controlled using Visual Basic for Applications (VBA). The basic framework is the same whether you're coding in Excel, Word, Access, Outlook, Powerpoint, Visio, or Projects. The main difference is that each application has it's own set of unique objects to work with. The majority of this guide will cover VBA in Excel, but feel free to explore the capabilities of the other office products.

1) We'll first want to add the `Developer Tab` to our ribbon. Open Excel and go to `File > Options > Customize Ribbon`, check off `Developer` and hit `OK`. You should now see a `Developer Tab` that you can use to open up the `Visual Basic` window. Note: You can also use hotkeys `Alt+F11` to open up the editor.

2) Next, we will want to configure our `Visual Basic` window to show some helpful tools and panels. This is mostly to preference so feel free to add and configure to whatever you're most comfortable with. Go to `View` and add the `Immediate Window` and `Properties Window`. You should already have the `Project Explorer` displayed with a tree structure of your workbook, but if you don't, you can add it by going to `View` and then `Project Explorer`. Finally, we're going to add the `Edit Toolbar` since it gives us some quick ways to format our code. Go to `View > Toolbars > Edit` and then drag it to the top where the rest of the toolbar lives.

Running your first script

VBA code can be stored in Application Objects (eg. Sheet1, Sheet2, Sheet3, ThisWorkbook), Userforms, Modules, or Class Modules. The most general place to add code is within a basic Module.

1) Go to `Insert > Module` and a new Module called 'Module1' should appear in your `Project Explorer` and open for editing. Let's change the name of this Module by using the `Properties Window` to change `(Name)` from 'Module1' to 'MyFirstScript'.

2) Now click in the main window and type in:

```
Sub HelloWorld()
  Debug.Print "Hello World!"
End Sub
```

3) Run your script by clicking on the green Play button in the toolbar or by hitting `F5`. Note that you will need to have your mouse cursor somewhere in the script you want to run. Otherwise, a pop-up will appear asking you to choose which script to run. You should see `Hello World!` get printed to your `Immediate Window`.

4) You can also step through your code line-by-line using `F8`. Give it a try!

Introduction to Data Variables

Data Variable Types

Note: These are just some of the most commonly used variables. For the full list of Data Varaible Types see Microsoft's Data Type Summary

---

- `String`: Denoted by double-quotes. "This is a string"
- `Integer`: Whole number between -32,768 and 32,767
- `Long`: Whole number between -2,147,483,648 and 2,147,483,647
- `Double`: Double-precision floating point
- `Boolean`: `TRUE` or `FALSE`
- `Date`: January 1, 100 to December 31, 9999
- `Variant`: Special variable that can hold any data type

Variable Declaration & Scope

There are three ways to declare a variable (each defining a different level of scope).

- `Dim`: Procedure level (local) variable. Must be declared within the procedure using it.
- `Private`: Module level variable. Visible to any procedure within the module. Must be declared at the top of the module.
- `Public`: Global level variable. Visible to any module within the project. Must be declared at the top of the module.

```
Public gMyPublicVar As Variant
Public mMyPrivateVar As Variant
Sub MyProcedure()
    Dim MyLocalVar As Variant
End Sub
```

Expanding on your first script (Pt. 1)

Let's now build upon our first script to apply what we've learned about variables and touch upon the basics of commenting, debugging, and user inputs.

1) Let's add a comment and a string variable to hold our message. Comments are initiated by a single quote. Unfortunately, the concept of Comment Blocks do not exist in VBA.

```
Sub HelloWorld()
  'This was my very first VBA script!
  Dim msg As String

  msg = "Hello World!"
```

(continues on next page)

(continued from previous page)

```
  Debug.Print msg
End Sub
```

2) Let's try out some debugging techniques.

   2.1) Click on the grey bar to the left of `Debug.Print msg` to add a breakpoint. Alternatively, click on that line and hit `F9`. Now run your script and it will stop right before executing that line of code (it will be highlighted yellow and nothing would have printed).

   2.2) In your `Immediate Window`, type in `?msg` and hit enter. This will return the value stored in your variable `msg`. You could also hover your mouse over `msg` in your script and a tooltip will appear showing it's value.

   2.3) Now in your `Immediate Window`, type `msg = "Bonjour World!"`. This reassigns the value stored in your variable. If you allow the script to finish executing by hitting Play or `F5`, it will print the new value we just assigned.

3) Let's now grab some info from our user to make our greeting a little more personalized. To do this, we'll also need to concatinate our strings together using `&`. Finally, we're also going to have the message pop up instead of print out.

```
Sub HelloWorld()
  'This was my very first VBA script!
  Dim msg As String
  Dim user As String

  user = Inputbox("What's your name?")
  msg = "Hello " & user & "!"
  MsgBox(msg)
End Sub
```

Introduction to Objects, Properties, and Methods

Objects

```
"Objects are the fundamental building block of Visual Basic;
nearly everything you do in Visual Basic involves modifying objects.
Every element of Microsoft Word - documents, tables, paragraphs, bookmarks,
fields, and so on - can be represented by an object in Visual Basic."
                                                    -Microsoft Dev Center
```

- An object can be a member of another object. For example, the Sheet Object is a member of the Workbook Object which is then a member of the Application Object. To access an Object's member, use a period (`Application.ThisWorkbook.ActiveSheet`).
  - In many cases, you don't need to explicitely define the full heirarchy down to the object you want work with. `Application.ActiveWorkbook.ActiveSheet.Cells(1,1).Value = "Hello World!"` is the fully defined heirarchy, but `Cells(1,1).Value = "Hello World!"` would work just the same.

Properties

```
"A property is an attribute of an object or an aspect of its behavior.
For example, properties of a document include its name, its content, and
its save status, as well as whether change tracking is turned on.  To change
the characteristics of an object, you change the values of its properties."
                                                        -Microsoft Dev Center
```

Methods

```
"A method is an action that an object can perform.  For example, just as a
document can be printed, the Document object has a PrintOut method. Methods
often have arguments that qualify how the action is performed."
                                                        -Microsoft Dev Center
```

Object Variables

Object variables allow you to store a reference to any object. The main difference in using an object variable as opposed to a data variable is that you need to use the word `Set` to assign something to it.

```
Sub MyProcedure()
    Dim xlSht As Excel.Worksheet
    Dim sheetName As String

    Set xlSht = ActiveSheet
    sheetName = xlSht.Name
End Sub
```

Note: This example uses early binding to declare the variable `xlSht` specifically as an `Excel.Worksheet` object. You could also use late binding to declare the variable as just an Object like `Dim xlSht As Object`. Early binding requires you to have the appropriate library references loaded beforehand.

Note: If you need help with any Object in VBA, your best resource is the `Object Browser`. Go to `View > Object Browser` or hit `F2` to open it up. The `Object Browser` allows you to look up anything about an Object including it's Properties and Methods.

Expanding on your first script (Pt. 2)

Let's build upon our first script one last time to practice using Objects, Properties, and Methods. First, we're going to read the user's name from the `Value Property` of the `Range Object` for Cell A1 and then we'll execute the object's `ClearContents Method` to clear out their name before displaying the message.

```
Sub HelloWorld()
  'This was my very first VBA script!
  Dim xlRng As Object
  Dim msg As String
  Dim user As String

  Set xlRng = ActiveSheet.Range("A1")
```

(continues on next page)

(continued from previous page)

```
  user = xlRng.Value
  xlRng.ClearContents
  msg = "Hello " & user & "!"
  MsgBox(msg)
End Sub
```

2.3.2 builtin - Class Modules

Class Modules allow you to create your own objects in VBA. Similar to built-in objects like the Workbook, Worksheet, or Range object, Class Module objects can have their own set of properties and methods.

There are four different items in a class module:

- Member Variables
- Properties
- Methods
- Events

To start, let's create a new Class Module in a workbook named `clsClient`. Let's also create a new regular Module so we can test things.

Member Variables

Member Variables are dimensioned using `Private` or `Public` within your Class Module. If they are dimensioned using `Public`, they can be accessed and manipulated from outside the Class Module.

```
'Class Module: clsClient
Public FullName AS String
```

```
'Regular Module
Sub TestMyClass()
    'Create Object from Class Module
    Dim oClient As New clsClient

    'Assign value to Public Member Variable
    oClient.FullName = "John Doe"

    'Retrieve value from Public Member Variable
    Debug.Print oClient.FullName
End Sub
```

It is usually best practice, however, to use `Private` Member Variables and then use Class Properties to set and get information.

Properties

The three commands for using properties in a Class Module are:

- Get: Returns an object or value
- Let: Sets a value
- Set: Sets an object

Let's turn our Member Variable "FullName" into a Class Property.

```
'Class Module: clsClient
Private msFullName As String

Public Property Get FullName() As String
    FullName = msFullName
End Property

Public Property Let FullName(ByVal sValue As String)
    msFullName = sValue
End Property
```

Your existing code in the regular class module will still work just the same, but if you step through it, you'll see how the property "FullName" is set when the value is assigned and retrieved when printed.

You can create ReadOnly properties by just using Get without Let.

```
'Class Module: clsClient
Private msFullName As String

Public Property Get FullName() As String
    FullName = msFullName
End Property

Public Property Let FullName(ByVal sValue As String)
    msFullName = sValue
End Property

Public Property Get FirstName() As String
    FirstName = Left(FullName, Instr(FullName, " ") - 1)
End Property

Public Property Get LastName() As String
    LastName = Right(FullName, Len(FullName) - Instr(FullName, " "))
End Property
```

```
'Regular Module
Sub TestMyClass()
    'Create Object from Class Module
    Dim oClient As New clsClient

    'Assign value to Public Member Variable
    oClient.FullName = "John Doe"

    'Retrieve value from Properties
    Debug.Print oClient.FullName
    Debug.Print oClient.FirstName
    Debug.Print oClient.LastName
End Sub
```

Methods

Class Methods are `Subs` or `Functions` in a Class Module.

```
'Class Module: clsClient
Private msFullName As String

Public Property Get FullName() As String
    FullName = msFullName
End Property

Public Property Let FullName(ByVal sValue As String)
    msFullName = sValue
End Property

Public Property Get FirstName() As String
    FirstName = Left(FullName, Instr(FullName, " ") - 1)
End Property

Public Property Get LastName() As String
    LastName = Right(FullName, Len(FullName) - Instr(FullName, " "))
End Property

Public Sub ExportToTextFile()
    Dim sFile As String

    sFile = Application.DefaultFilePath & "\client.txt"
    Open sFile For Output As #1
        Write #1, "First: " & FirstName
        Write #1, "Last: " & LastName
    Close #1
    MsgBox "Exported to " & sFile & "!"
End Sub
```

```
'Regular Module
Sub TestMyClass()
    'Create Object from Class Module
    Dim oClient As New clsClient

    'Assign value to Public Member Variable
    oClient.FullName = "John Doe"

    'Export Client to Text File
    oClient.ExportToTextFile
End Sub
```

Events

A Class Module has two events:

- Initialize: Triggered when new object of class is created
- Terminated: Triggered when class object is deleted

In other programming languages, these may be referred to as the `Constructor` and the `Destructor`. However, you cannot pass parameters to `Initialize` like you would a `Constructor`.

Add these to the bottom of your class module code:

```
Private Sub Class_Initialize()
    MsgBox "Class Initialized"
```

(continues on next page)

(continued from previous page)

```
End Sub

Private Sub Class_Terminate()
    MsgBox "Class Terminated"
End Sub
```

2.3.3 builtin - Enumerations

Enumerations are constants which represent a set of possible values. Microsoft has a number of built-in enumerations so that we developers can remember the names instead of numbers. The following are all examples of built-in enumerations:

- xlCalculationManual
- xlCalculationAutomatic
- xlCalculationSemiautomatic
- vbYesNo
- vbOKOnly

You can use your immediate window to see the value behind an enumeration. `?xlCalculationManual` will print out `-4135`.

Enum Statement

You can define your own enumeration set with the `Enum` statement. By default, the enumerated values will start at 0 and increment upwards, but you can override this with your own values. As an example, say you want to take some action depending on what button a user presses on a MsgBox. The MsgBox, when used as a function, will return an integer, but we can define an enumeration set to use instead.

```
Public Enum vbMsgBoxResult
    vbOK = 1
    vbCancel = 2
    vbAbort = 3
    vbRetry = 4
    vbIgnore = 5
    vbYes = 6
    vbNo = 7
End Enum
Sub WithoutEnum()
    Select Case Msgbox("Click Yes or No", vbYesNo)
        Case 6
            Msgbox "You clicked Yes!"
        Case 7
            Msgbox "You clicked No!"
    End Select
End Sub
Sub WithEnum()
    Select Case Msgbox("Click Yes or No", vbYesNo)
        Case vbYes
            Msgbox "You clicked Yes!"
        Case vbNo
            Msgbox "You clicked No!"
```

(continues on next page)

(continued from previous page)

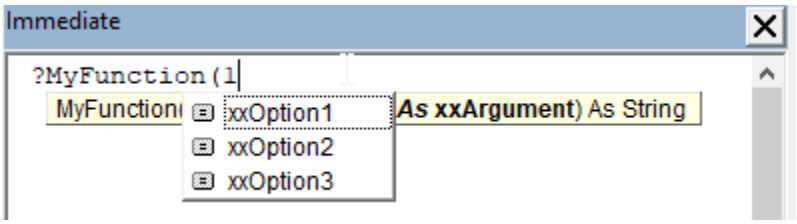
```
    End Select
End Sub
```

Enumerations as Arguments

You may have noticed by now that visual basic will help you auto-complete your code by displaying available options. For example, if you type `Application.Calculation=`, you should see the list of calculation options you can use. We can do the same by creating an enumeration set and then defining it as an input to a sub or function.

```
Public Enum xxArgument
    xxOption1 = 1
    xxOption2
    xxOption3
End Enum
Function MyFunction(ByVal InputOption As xxArgument) As String
    MyFunction = "You used option " & InputOption
End Function
```

Put the above code into a new module and then type `?MyFunction(` into your immediate window. It should display the three options we created as potential inputs to help auto-complete! See figure below:



2.3.4 builtin - Runtime Errors

By default, VBA will throw a dialog box with an error number and description when it encounters a runtime error. The user is asked to either end the code execution or debug the error themselves by going to the line where it broke. This behavior can be changed using the command `On Error`.

- `On Error GoTo [Line]` will cause the code to jump to a specific line.
- `On Error Resume Next` will force the code to continue running even if an error is encountered.
- `On Error GoTo 0` will reset the error handling behavior back to default.

The example below shows the general structure of a procedure with an error handler. If you run the procedure and enter a string (such as "hello world") when prompted for a number, our own error prompt will appear and then the code will exit.

```
Sub OnErrorGoToLine()
    Dim dVar As Double

    On Error GoTo ErrLine
    dVar = InputBox("Enter a number")

    MsgBox "Good job!"
ExitLine:
    On Error GoTo 0
```

(continues on next page)

(continued from previous page)

```
    Exit Sub
ErrLine:
    MsgBox "That is not a number!"
    Err.Clear
    Resume ExitLine
End Sub
```

Validation Errors

You can use a similar structure and the `GoTo` command for handling errors caused by validation checks within your code. These are things that wouldn't trigger a runtime error, but maybe you wouldn't want to allow the rest of the code to execute if certain conditions aren't met. Note that you have to use `GoTo` instead of `Resume` in your handler (`Resume` only applies to runtime errors).

```
Sub ValidationErrors()
    Dim vVar As Variant

    vVar = Inputbox("Enter an even positive whole number no larger than 10")

    If Len(vVar) = 0 Then GoTo ErrNothingEntered
    If Not IsNumeric(vVar) Then GoTo ErrNotNumeric
    If Not Int(vVar) = vVar Then GoTo ErrNotInteger
    If Not vVar/2 = Int(vVar/2) Then GoTo ErrNotEven
    If vVar <= 0 Then GoTo ErrNotPositive
    If vVar > 10 Then GoTo ErrTooLarge

    MsgBox "Good job!"
ExitLine:
    Exit Sub
ErrNothingEntered:
    MsgBox "You didn't enter anything..."
    GoTo ExitLine
ErrNotNumeric:
    MsgBox "That isn't a number..."
    GoTo ExitLine
ErrNotInteger:
    MsgBox "That isn't a whole number..."
    GoTo ExitLine
ErrNotEven
    MsgBox "That is not an even number..."
    GoTo ExitLine
ErrNotPositive
    MsgBox "That is not a positive number..."
    GoTo ExitLine
ErrTooLarge:
    MsgBox "That is larger than 10..."
    GoTo ExitLine
End Sub
```

Clean Ups

One of the main reasons we want to have an error handler is to clean up our environment before allowing the code to exit. For example, say we have some code that is manipulating a spreadsheet so we decide to turn off calculations

and events to gain speed. If our code breaks without an error handler and the user ends execution, those settings will remain off. We can use the `ExitLine` of our code to house our clean up items so this doesn't happen.

```
Sub ErrorWithCleanUp()

    With Application
        .Calculation = xlCalculationManual
        .EnableEvents = False
        .ScreenUpdating = False
    End With

    On Error GoTo ErrLine
    'Some code that does stuff

ExitLine:
    On Error GoTo 0
    With Application
        .Calculation = xlCalculationAutomatic
        .EnableEvents = True
        .ScreenUpdating = True
    End With
    Exit Sub
ErrLine:
    MsgBox Err.Number & ": " & Err.Description
    Err.Clear
    Resume ExitLine
End Sub
```

It's also good practice to close any hidden objects and release object variables from memory.

```
Sub ErrorReleaseObjects()
    Dim xlApp As Object
    Dim xlWb As Object

    On Error GoTo ErrLine
    Set xlApp = CreateObject("Excel.Application")
    Set xlWb = xlApp.Workbooks.Open("C:\SomeRandomSpreadsheet.xlsx")

    'Some code that does stuff

ExitLine:
    On Error GoTo 0
    If Not xlWb Is Nothing Then
        xlWb.Saved = True
        xlWb.Close
        Set xlWb = Nothing
    End If
    If Not xlApp Is Nothing Then
        xlApp.Quit
        Set xlApp = Nothing
    End If
    Exit Sub
ErrLine:
    MsgBox Err.Number & ": " & Err.Description
    Err.Clear
    Resume ExitLine
End Sub
```

2.3.5 builtin - Events

Events allow you to have code triggered by specific actions. In Excel, these could be:

- Sheet Level Events
  - Clicking or double-clicking on a cell
  - Changing a cell's value
  - Calculating the sheet
- Workbook Level Events
  - Adding or changing a sheet
  - Opening, saving, or closing a workbook
- UserForm Level Events
  - Initializing a userform
  - Clicking a button on a userform

Turning Events On/Off

Events can have a negative impact on user experience if they are triggered too often and/or have long runtimes. You could also find yourself writing an event that triggers itself or others unintentionally. For example, if you had the following `Worksheet_Change` event in your workbook, changing the value of any cell would cause an infinite loop caused by the change event changing the value of the cell below and re-triggering itself.

```
Private Sub Worksheet_Change(ByVal Target As Range)
    Target.Offset(1,0).Value = Target.Value
End Sub
```

If you ever need to run some code without having an event fire you would use `Application.EnableEvents = False`. Don't forget to turn events back on after your code runs using `Application.EnableEvents = True`. Also, avoid the common pitfall of turning events off and not have an error handler to turn them back on when things go wrong!

Event Optimization

It's sometimes worthwhile to have kickout logic at the beginning of your event to handle cases where you don't want your code to run. For example, say we only want to run some code if the user enters something into a single cell. We would want to kick out if the user changes more than one cell or if the post-change value was null.

```
Private Sub Worksheet_Change(ByVal Target As Range)
    If Target.Count > 1 Then Exit Sub
    If Target.Value = "" Then Exit Sub
    'Code to run
End Sub
```

2.3.6 builtin- Interapplication Control

One of the best things about VBA is that it's the common language used by all Microsoft Office Applications. If you master VBA in Excel, you could easily learn VBA for Access, Word, Outlook, Powerpoint, Visio, or Projects. They each have their own libraries and differ only in terms of the Objects being manipulated, but the core language

and coding structure is exactly the same. This also makes controlling one Office Application from another incredibly trivial.

Early Binding vs. Late Binding

The process of assigning an object to an object variable is called "binding". In Early Binding (Static Binding), this occurs during compile time. In Late Binding (Dynamic Binding) , this doesn't happen until runtime.

The benefit of Early Binding in VBA is that it'll ensure that you're using the proper Objects, Properties, and Methods as you code (ie. it will provide auto-complete options, auto-check syntax, and allows you to use built-in enumerations). The downside of Early Binding in VBAis that you will need to add appropriate library references in order for the code to compile at all. To add a reference,go to Tools > References and then check off the reference you want to add. In the example below, you will need to add the `Microsoft Outlook X.X Object Library` to your workbook before the code can work.

```
Sub EarlyBindingExample()
    Dim olApp As Outlook.Application
    Dim olMsg As Outlook.MailItem

    Set olApp = CreateObject("Outlook.Application")
    Set olMsg = olApp.CreateItem(olMailItem)
        olMsg.Display
End Sub
```

By contrast, the benefit of Late Binding is that you don't have to worry about having Library References in order for your code to work. This becomes extremely helpful when you're building reusable code that may end up in many workbooks. Late Binding allows you to just copy and paste the code in that new workbook and have it work right away! The downside of Late Binding is that you don't get the help of the auto-complete, auto-checker, and must use values instead of the enumerations.

```
Sub LateBindingExample()
    Dim olApp As Object
    Dim olMsg As Object

    Set olApp = CreateObject("Outlook.Application")
    Set olMsg = olApp.CreateItem(0)
        olMsg.Display
End sub
```

A happy medium might be to code using Early Binding and then replace your specific object variable references with just `Object` and then replace your enumerations with it's value before removing the library reference. You can check the value of an enumeration by typing in "?" followed by the enumeration in your immediate window (`?olMailItem` = 0).

Setting An Application Instance

The first thing you need to do to control another application is grab an instance of it. There are two ways to do this:

- Set olApp = CreateObject("Outlook.Application")
- Set olApp = GetObject(, "Outlook.Application")

As the syntax suggests, the first method will create a new instance while the second will try to grab an existing instance. The second method is useful if you're trying to access something that's already open (a workbook for example), but comes with the risk of throwing an error if there is no existing instance to grab. Usually, the first method is preferred because it allows you to control an instance without impacting what the user is doing in their existing instance.

Note that the first method will create an instance that is hidden in the background. To make it visible, set the Application's Visible property = True. Also, most application instances will remain open even after the code finishes executing (the exception is Outlook). If your code is doing something to a Word document or Excel spreadsheet and you don't set it's visibility to True and don't quit the application in the end, your user may end up with a number of open application instances running in the background.

Once you have the Application instance, you can access all of the Objects within that Application by drilling down into it's members!

Example #1: Sending an Outlook Email

This code can be used from any Office Application to create an email in Outlook.

```
Sub CreateEmail(ByVal sSubject As String, _
                ByVal sHTMLBody As String, _
                ByVal sTo As String, _
                Optional ByVal sCC As String = "", _
                Optional ByVal sBC As String = "", _
                Optionl ByVal bSend As Boolean = False)
    Dim olApp As Object
    Dim olMsg As Object

    Set olApp = CreateObject("Outlook.Application")
    Set olMsg = olApp.CreateItem(0)

    With olMsg
        .Subject = sSubject
        .htmlBody = sHTMLBody
        .To = sTo
        .CC = sCC
        .BC = sBC
        If bSend Then
            .Send
        Else
            .Display
        End If
    End With

ExitLine:
    Set olMsg = Nothing
    set olApp = Nothing
End Sub
```

Example #2: Exporting an Access Table or Query to Excel

This code can be used from an Access Database to export the contents of a table to query into an Excel spreadsheet. Note: This is not the only way to export data from Access to Excel!

```
Sub ExportData(ByVal sTableOrQuery As String)
    Dim xlApp As Object
    Dim xlWb As Object
    Dim rst As Recordset
    Dim fld As Field
    Dim iFld As Integer
```

(continues on next page)

(continued from previous page)

```
    Set rst = DBEngine(0)(0).OpenRecordset(sTableOrQuery, dbOpenSnapshot)
    Set xlApp = CreateObject("Excel.Application")
        xlApp.Visible = True
    Set xlWb = xlApp.Workbooks.Add
        With xlWb.Sheets(1)
            For Each fld In rst.Fields
                iFld = iFld + 1
                .Cells(1, iFld).Value = fld.Name
            Next
            .Cells(2,1).CopyFromRecordset rst
        End with

ExitLine:
    rst.Close
    Set rst = Nothing
    Set xlWb = Nothing
    Set xlApp = Nothing
End Sub
```

2.3.7 builtin - Logic Loops

True False

In VBA, the following all evaluate to True:

- -1=True
- 0=False

If ElseIf Else

```
Sub IfElseIfElse()
    x = Inputbox("Enter a number between 1 and 10")
    If x < 5 Then
        MsgBox "x was less than 5"
    ElseIf x < 7 Then
        MsgBox "x was greater than or equal to 5 but less than 7"
    Else
        MsgBox "x was greater than or equal to 7"
    End If
End Sub
```

Select Case

```
Sub SelectCase()
    x = Inputbox("Enter either 'Cat', 'Dog', 'Bush', or 'Tree'")
    Select Case x
        Case "Cat", "Dog"
            MsgBox "That is an animal"
        Case "Bush", "Tree"
            MsgBox "That is a plant"
        Case Else
```

(continues on next page)

(continued from previous page)

```
            MsgBox "I do not know what that is"
    End Select
End Sub
```

For Loops

```
Sub ForLoops()
    'Basic Incremental Loop
    For i = 0 To 10
        Debug.Print i
    Next

    'Custom Step Size Loop
    For i = 0 To 10 Step 2
        Debug.Print i
    Next

    'Reverse Loop
    For i = 10 To 0 Step -1
        Debug.Print i
    Next

    'For Each Loop
    For Each xlSht In ThisWorkbook.Sheets
        Debug.Print xlSht.Name
    Next

    'Exiting For Loop Early
    For i = 0 To 10
        Debug.Print i
        If i = 5 Then
            Exit For
        End If
    Next
End Sub
```

Do Loops

There are two forms of the Do While/Do Until Loops. The difference is that one will evaluate the criteria before entering the loop while the other won't evaluate the criteria until at the end of the loop. In the example script below, try changing the value of x at the start of each loop to 5 and see what happens.

```
Sub DoLoops()
    'Do Until Loop #1
    x = 1
    Do Until x = 5
        x = x + 1
        Debug.Print x
    Loop

    'Do Until Loop #2
    x = 1
    Do
```

(continues on next page)

(continued from previous page)

```
        x = x + 1
        Debug.Print x
    Loop Until x = 5

    'Do While Loop #1
    x = 1
    Do While x < 5
        x = x + 1
        Debug.Print x
    Loop

    'Do While Loop #2
    x = 1
    Do
        x = x + 1
        Debug.Print x
    Loop While x < 5
End Sub
```

Looping Through Files in a Folder

This is a method of looping through files in a folder using the `Dir()` function.

```
Sub FileLoop()
    Dim MyFile As String

    'Looping through all files
    MyFile = Dir("C:\", vbNormal)
    Do While Len(MyFile) > 0
        Debug.Print MyFile
        MyFile = Dir()
    Loop

    'Looping through .csv files
    MyFile = Dir("C:\*.csv", vbNormal)
    Do While Len(MyFile) > 0
        Debug.Print MyFile
        MyFile = Dir()
    Loop
End Sub
```

2.3.8 builtin - Subroutines and Functions

The main difference between a subroutine and function is that a function returns a value while a subroutine does not. When calling one subroutine from another, you would use the `Call` command (Note: the code can run without using `Call`, but it's much easier to understand what's happening when its there). To use a function you created, just assign it to a variable like any other function.

```
Sub MyFirstSub()
    Call MySecondSub
End Sub
Sub MySecondSub()
    x = MyFunction()
```

(continues on next page)

(continued from previous page)

```
    Debug.Print x
End Sub
Function MyFunction() As String
    MyFunction = "Hello World!"
End Function
```

Input Arguments

Both subroutines and functions can take in any number of arguments. These can either be passed by value (`ByVal`) or by reference (`ByRef`). `ByVal` will send the value of the variable while `ByRef` will send the variable itself. In both cases, you should also define the type of variable being passed.

```
Sub Main()
    Dim x As Integer
    x = 3
    Debug.Print TripleByVal(x)  ' this return 3*3=9
    Debug.Print x               ' this returns the original x=3
    Debug.Print TripleByRef(x)  ' this return 3*3=9, but...
    Debug.Print x               ' it ByRef also returns any changes to variable, hence␣
↪x=9
End Sub
Function TripleByVal(ByVal x As Integer) AS Integer
    x = x * 3
    TripleByVal = x
End Function
Function TripleByRef(ByRef x As Integer) AS Integer
    x = x * 3
    TripleByRef = x
End Function
```

If you run the above `Main()` subroutine, you should see the following numbers printed to your immediate window: 9, 3, 9, 9. You can see how passing `ByRef` passed the variable itself so it retains any changes made to it. By default, VBA will pass arguments `ByRef` so be sure to use `ByVal` if you don't want that behavior.

Optional Arguments

You can define arguments as optional if you don't always need it. It is usually a good idea to define a default value to use if the argument is ommited. If you don't, VBA will apply it's own default value (usually an empty string or 0).

```
Sub Main()
    MsgBox MyGreeting("John")
    MsgBox MyGreeting()
End Sub
Function MyGreeting(Optional ByVal UserName As String = "Human") As String
    MyGreeting = "Hello " & UserName & "!"
End Function
```

2.3.9 Tips and Tricks

This section is a collection of tips and tricks to help make your code more stable, efficient, or supportable.

Named Ranges

If you reference a range from code, it is highly advised that you use a named range instead of the range address.

- You can create a named range by selecting the range and then editing the area to the left of the formula bar
- Or by going to `Formulas > Name Manager`.

Using named ranges will make your code much more stable as they will move around when changes are made to your worksheet. For example, say your code was grabbing a name from cell A1. Now imagine you or another user adds a new column or row above or to the left of A1. Without named ranges, your code would still be looking for something in A1, but that cell has now moved to A2 or B1. With named ranges, your code would work without issue.

```
' code reference by address (not desirable)
Sub UsingRangeAddress()
    Debug.Print Range("A1").Value
End Sub

' code reference by namespace (preferred)
Sub UsingNamedRanges()
    Debug.Print Range("UserName").Value
End Sub
```

Calculations, Events, and Screen Updating

If your code is manipulating your spreadsheet and you find it taking a long time to run, try turning off calculations, events, and screen updating. Don't forget to turn them back on at the end!

```
Sub MyProcedure()
    With Application
        .Calculation = xlCalculationManual
        .EnableEvents = False
        .ScreenUpdating = False
    End With

    'Code to run

    With Application
        .Calculation = xlCalculationAutomatic
        .EnableEvents = True
        .ScreenUpdating = True
    End With
End Sub
```

R1C1 Reference Style

R1C1 is great for writing chunks of formula along a row or column. You can switch between viewing your formulas in R1C1 by going to `File > Options > Formulas > R1C1 Reference Style`. You do not need to have this option on to write formulas in R1C1.

For an example, say we want to write formulas in cells A2:Z2 to make them equal to the cell above them.

```
Sub FormulasWithoutR1C1()
    For i = 1 To 26
        Cells(2, i).Formula = "=" &  Split(Cells(1, i).Address, "$")(1) & 1
    Next
```

(continues on next page)

(continued from previous page)

```
End Sub
Sub FormulasWithR1C1()
    Range("A2:Z2").FormulaR1C1 = "=R[-1]C"
End Sub
```

2.3.10 access - Database Management Systems

Access is Microsoft's database product. It's important to note that, at it's core, Access is a Database Management System (DBMS). The application is controlled via Visual Basic, but the standard DBMS language is SQL. The GUI of Access allows users to operate it without knowledge of SQL, but it is highly recommended that you learn SQL if you plan on controlling Access with VBA. There are a few variations of SQL to be aware of. The SQL used in Access has a few differences in syntax from the more robust T-SQL. Below are a few examples:

Delimiters

- Text: Access accepts both `"` and `'` while T-SQL only accepts `'`
- Date: Access uses `#` while T-SQL uses `'`
- Wildcard: Access uses `*` while T-SQL uses `%`

Functions

- Conditions: Access uses `IIF()` while T-SQL uses `CASE, IF-THEN-ELSE`
- Conversions: Access uses VBA functions like `CDate()`, `CLng()`, `CInt()` while T-SQL uses `CAST()`
- Nulls: Access uses `Nz()` while T-SQL uses `ISNULL()`

Access Objects

There are four main Access Objects.

- Tables
- Queries
- Forms
- Reports

Tables and Queries are similar in that they both represent data. Forms and Reports are similar in that they both represent a UI. As such, Tables and Queries share similar properties and methods as do Forms and Reports. For this tutorial, we will only be focusing on data retrieval and manipulation (Queries).

Queries are just stored pieces of SQL. When a query is opened, it presents the user with the results of the SQL execution. The query builder UI in Access allows users to drag and drop tables/queries, create joins, select fields, apply criteria, and sort their data. All of these interactions with the UI are translated into SQL behind the scenes. You can create a query in the UI, and then go to `SQL View` to see the resulting SQL. The main thing to remember is that anything you can do with a query, you can do by executing SQL. Therefore, any automation of data in Access using VBA involves generating SQL strings and executing them.

For the rest of the tutorial, we'll be using the following `Customers` table dataset:

ID	Name	State	Date	Sale
1	John Doe	WA	1/1/2019	500.50
2	Jane Smith	OR	2/1/2019	250.25
3	Mike White	CA	3/1/2019	1000.00
4	Mike Smith	OR	4/1/2019	300.00

Recordsets

The main object that you'll use in Access VBA are recordsets. These are digital representations of data that you can use to iterate through, add/modify records, or grab values to use in variables for the rest of your code.

A recordset is created using the `OpenRecordset` method of the `Database` object. You can open a recordset as ReadOnly using `dbOpenSnapshot`. If you plan to make edits to the data, use `dbOpenDynaset`.

Some important properties and methods for recordsets are listed below:

- `BOF`: Beginning of file (Boolean Property)
- `EOF`: End of file (Boolean Property)
- `Edit`: Edit record (Method)
- `AddNew`: Add new record (Method)
- `Update`: Commit changes to record (Method)
- `Delete`: Delete current record (method)
- `MoveNext`: Move pointer to next record (Method)
- `MoveFirst`: Move pointer to first record (Method)
- `Close`: Close recordset connection (Method)

This example creates a recordset of Customers who live in either WA or OR and prints their names to the immediate window. Note: A recordset that is both at the beginning of a file and the end of a file simultaneously is empty.

```
Sub LoopThroughRecordset()
    Dim rst As Recordset
    Dim sql As String

    sql = "SELECT Name FROM Customers WHERE State In('WA','OR');"
    Set rst = DBEngine(0)(0).OpenRecordset(sql, dbOpenSnapshot)

    With rst
        If Not .BOF And Not .EOF Then
            Do
                Debug.Print .Fields("Name").Value
                .MoveNext
            Loop Until .EOF
        End If
    End With
ExitLine:
    rst.Close
    Set rst = Nothing
End Sub
```

This example uses a recordset to add a new customer to the table and then prints their new ID (ID must be an AutoNumber to work!).

```
Sub AddCustomerRecordset()
    Dim rst As Recordset
    Dim sql As String
    Dim iID AS Long

    sql = "SELECT * FROM Customers;"
    Set rst = DBEngine(0)(0).OpenRecordset(sql, dbOpenDynaset)

    With rst
        .AddNew
        iID = .Fields("ID").Value
        .Fields("Name").Value = "Daniel Park"
        .Fields("State").Value = "CA"
        .Fields("Date").Value = CDate("5/1/2019")
        .Fields("Sale").Value = 777.77
        .Update
    End With
    Debug.Print iID
ExitLine:
    rst.Close
    Set rst = Nothing
End Sub
```

This example uses a recordset to update the state of anyone in CA to HI.

```
Sub UpdateStateRecordset()
    Dim rst As Recordset
    Dim sql As String

    sql = "SELECT * FROM Customers;"
    Set rst = DBEngine(0)(0).OpenRecordset(sql, dbOpenDynaset)

    With rst
        If Not .BOF And Not .EOF Then
            Do
                If .Fields("State") = "CA" Then
                    .Edit
                    .Fields("State") = "HI"
                    .Update
                End If
                .MoveNext
            Loop Until .EOF
        End If
    End With
ExitLine:
    rst.Close
    Set rst = Nothing
End Sub
```

DoCmd

If you're familiar with SQL, you might've noticed that the previous two examples are actually pretty inefficient for what they're doing. An similar thing could be accomplished using a single SQL statement representing an `Action Query`.

The `DoCmd` class has a number of useful methods that can be used to automate behavior in Access. One of these meth-

ods is the `DoCmd.RunSQL` method. Below is the last example to update States recreated using `DoCmd.RunSQL`.

```
Sub UpdateStateRunSQL()
    Dim sql As String

    sql = "UPDATE Customers SET State = 'HI' WHERE State = 'CA';"
    DoCmd.RunSQL sql
End Sub
```

If you run this, you may notice a pop-up asking for confirmation on the change you're about to make. To suppress this, we can use the `DoCmd.SetWarnings` method.

```
Sub UpdateStateRunSQL()
    Dim sql As String

    sql = "UPDATE Customers SET State = 'HI' WHERE State = 'CA';"
    DoCmd.SetWarnings False
    DoCmd.RunSQL sql
    DoCmd.SetWarnings True
End Sub
```

There are many more methods of `DoCmd`. Check them out using the Object Browser!

DFunctions

Access has a few functions to look up and calculate statistics on data.

- `DLookup()`: Similar to `VLookup()` in Excel, but allows for multiple criteria
- `DMin()`: Similar to `Min()` in Excel, but allows for multiple criteria
- `DMax()`: Similar to `Max()` in Excel, but allows for multiple criteria
- `DCount()`: Similar to `CountIfs()` in Excel
- `DSum()`: Similar to `SumIfs()` in Excel

All of these functions have the same three arguments:

1. FieldName
2. TableName or QueryName
3. Criteria

The example below uses `DLookup()` to get the first ID of a customer named Mike who lives in OR. It's important to also use the `Nz()` function to handle nulls if no record matches our criteria. We'll just run this in the immediate window.

```
?Nz(DLookup("ID", "Customers", "Name = 'Mike*' AND State = 'OR'"),0)
```

This example uses `DSum()` to calculate the total sales in CA. `DCount()` and `DSum` do not need an `Nz()` wrapper.

```
?DSum("Sale", "Customers", "State = 'CA'")
```

We can now recreate the second recordset example of adding a customer using `DoCmd.RunSQL` to append the record and `DMax()` to get the newly added ID.

```
Sub AddCustomerRunSQL()
    Dim sql As String
    Dim iID AS Long

    sql = "INSERT INTO Customers ( Name, State, Date, Sale ) " & _
            "SELECT 'Daniel Park' AS Name, 'CA' As State, #5/1/2019# As Date, 777.77␣
↪As Sale;"

    DoCmd.SetWarnings False
    DoCmd.RunSQL sql
    DoCmd.SetWarnings True

    iID = DMax("ID", "Customers")
    Debug.Print iID
End Sub
```

Access from Excel

Below is a custom class module for Excel that allows you to interface and manipulate data stored in an Access Database or SQL Server using Access-like syntax. To use it, copy the code into a new class module and name it `clsDB`. You will also need to add a reference to `Microsoft Office X.X Access Database Engine Object Library`.

```
'Class Module: clsDB
'Author: Kevin Kim
'Required Reference: Microsoft Office X.X Access Database Engine Object Library

Private Enum dbCnnType
    dbCnnTypeAccess
    dbCnnTypeSQLServer
End Enum
Private mCnnType As Integer
Private mTempDB As DAO.Database
Private sConnection As String
Private sDB As String
Public Property Get Connection() As String
    Connection = sConnection
End Property
Public Property Get CnnType() As Integer
    CnnType = mCnnType
End Property
Public Property Let CnnType(aValue As Integer)
    mCnnType = aValue
End Property
Public Property Let Connection(aValue As String)
    If Len(Dir(aValue, vbNormal)) > 0 Then
        sDB = aValue
        CnnType = dbCnnTypeAccess
        sConnection = vbNullString
    Else
        CnnType = dbCnnTypeSQLServer
        sConnection = aValue
    End If
End Property
Public Function SQLServerConnection(ServerName As String, Database As String) As␣
↪String
```

(continues on next page)

(continued from previous page)

```
    SQLServerConnection = "ODBC;Driver={SQL Server};" & _
                                "Server=" & ServerName & ";" & _
                                "Database=" & Database & ";"
End Function
Private Function TempDB() As DAO.Database
    Dim oWS As DAO.Workspace
    Dim sTempDB As String

    If mTempDB Is Nothing Then
        Set oWS = DBEngine.Workspaces(0)

        If CnnType = dbCnnTypeAccess Then
            sTempDB = sDB
        ElseIf CnnType = dbCnnTypeSQLServer Then
            sTempDB = Environ("Temp") & "\temp.accdb"

            If Len(Dir(sTempDB)) > 0 Then
                Kill sTempDB
            End If

            oWS.CreateDatabase sTempDB, dbLangGeneral
        End If

        Set mTempDB = oWS.OpenDatabase(sTempDB)
    End If

ExitLine:
    Set TempDB = mTempDB
    Exit Function
End Function
Public Function OpenRecordSet(sql As String, _
                                Optional RecordsetType As Integer = dbOpenSnapshot)␣
↪As DAO.Recordset
    Dim qdef As DAO.QueryDef

    Set qdef = TempDB.CreateQueryDef(vbNullString)

    If Connection <> vbNullString Then
        qdef.Connect = Connection
    End If

    With qdef
        .sql = sql
        .ReturnsRecords = True
        Set OpenRecordSet = .OpenRecordSet(RecordsetType)
    End With

ExitLine:
    Set qdef = Nothing
    Exit Function
End Function
Public Sub RunSQL(sql As String)
    Dim qdef As DAO.QueryDef

    Set qdef = TempDB.CreateQueryDef(vbNullString)

    If Connection <> vbNullString Then
```

(continues on next page)

(continued from previous page)

```
        qdef.Connect = Connection
    End If

    With qdef
        .sql = sql
        .ReturnsRecords = False
        .Execute (dbSeeChanges)
    End With

ExitLine:
    Set qdef = Nothing
    Exit Sub
End Sub
Public Function QueryDef(Item As Variant) As DAO.QueryDef
    Set QueryDef = TempDB.QueryDefs(Item)
End Function
Public Function TableDef(Item As Variant) As DAO.TableDef
    Set TableDef = TempDB.TableDefs(Item)
End Function
Public Function DLookup(Expr As String, _
                        Domain As String, _
                        Optional Criteria As String = vbNullString) As Variant
    Dim rst As DAO.Recordset
    Dim sql As String

    sql = "SELECT TOP 1 " & Expr & " As MyVal " & _
          "FROM " & Domain
        If Criteria <> vbNullString Then
            sql = sql & " " & _
                  "WHERE " & Criteria
        End If
        sql = sql & ";"

    Set rst = OpenRecordSet(sql)
        If Not rst.BOF And Not rst.EOF Then
            DLookup = rst(0)
        Else
            DLookup = Null
        End If

ExitLine:
    rst.Close
    Set rst = Nothing
End Function
Public Function DSum(Expr As String, _
                        Domain As String, _
                        Optional Criteria As String = vbNullString) As Variant
    Dim rst As DAO.Recordset
    Dim sql As String

    sql = "SELECT SUM(" & Expr & ") AS MyVal " & _
          "FROM " & Domain
        If Criteria <> vbNullString Then
            sql = sql & " " & _
                  "WHERE " & Criteria
        End If
        sql = sql & ";"
```

(continues on next page)

(continued from previous page)

```
    Set rst = OpenRecordSet(sql)
        If Not rst.BOF And Not rst.EOF Then
            DSum = rst(0)
        Else
            DSum = 0
        End If

ExitLine:
    rst.Close
    Set rst = Nothing
End Function
Public Function DCount(Expr As String, _
                       Domain As String, _
                       Optional Criteria As String = vbNullString) As Variant
    Dim rst As DAO.Recordset
    Dim sql As String

    sql = "SELECT Count(" & Expr & ") AS MyVal " & _
            "FROM " & Domain
        If Criteria <> vbNullString Then
            sql = sql & " " & _
                    "WHERE " & Criteria
        End If
        sql = sql & ";"

    Set rst = OpenRecordSet(sql)
        If Not rst.BOF And Not rst.EOF Then
            DCount = rst(0)
        Else
            DCount = 0
        End If

ExitLine:
    rst.Close
    Set rst = Nothing
End Function
Public Function DMax(Expr As String, _
                       Domain As String, _
                       Optional Criteria As String = vbNullString) As Variant
    Dim rst As DAO.Recordset
    Dim sql As String

    sql = "SELECT Max(" & Expr & ") AS MyVal " & _
            "FROM " & Domain
        If Criteria <> vbNullString Then
            sql = sql & " " & _
                    "WHERE " & Criteria
        End If
        sql = sql & ";"

    Set rst = OpenRecordSet(sql)
        If Not rst.BOF And Not rst.EOF Then
            DMax = rst(0)
        Else
            DMax = Null
        End If
```

(continues on next page)

(continued from previous page)

```
ExitLine:
    rst.Close
    Set rst = Nothing
End Function
Public Function DMin(Expr As String, _
                     Domain As String, _
                     Optional Criteria As String = vbNullString) As Variant
    Dim rst As DAO.Recordset
    Dim sql As String

    sql = "SELECT Min(" & Expr & ") AS MyVal " & _
          "FROM " & Domain
        If Criteria <> vbNullString Then
            sql = sql & " " & _
                  "WHERE " & Criteria
        End If
        sql = sql & ";"

    Set rst = OpenRecordSet(sql)
        If Not rst.BOF And Not rst.EOF Then
            DMin = rst(0)
        Else
            DMin = Null
        End If

ExitLine:
    rst.Close
    Set rst = Nothing
End Function
Public Function ObjectExists(sObjectType As String, sObjectName As String) As Boolean
     Dim tbl As DAO.TableDef
     Dim qry As DAO.QueryDef
     Dim i As Integer

     If sObjectType = "Table" Then
          For Each tbl In TempDB.TableDefs
               If tbl.Name = sObjectName Then
                    ObjectExists = True
                    Exit Function
               End If
          Next tbl
     ElseIf sObjectType = "Query" Then
          For Each qry In TempDB.QueryDefs
               If qry.Name = sObjectName Then
                    ObjectExists = True
                    Exit Function
               End If
          Next qry
     ElseIf sObjectType = "Form" Or sObjectType = "Report" Or sObjectType = "Module"␣
↪Then
          For i = 0 To TempDB.Containers(sObjectType & "s").Documents.Count - 1
               If DB.Containers(sObjectType & "s").Documents(i).Name = sObjectName␣
↪Then
                    ObjectExists = True
                    Exit Function
               End If
```

(continues on next page)

(continued from previous page)

```
            Next i
        ElseIf sObjectType = "Macro" Then
            For i = 0 To TempDB.Containers("Scripts").Documents.Count - 1
                If DB.Containers("Scripts").Documents(i).Name = sObjectName Then
                    ObjectExists = True
                    Exit Function
                End If
            Next i
        Else
            MsgBox "Invalid Object Type passed, must be Table, Query, Form, Report,␣
↪Macro, or Module"
        End If

End Function
Public Function Nz(aValue As Variant, aValueIfNull As Variant) As Variant
    If IsNull(aValue) Then
        Nz = aValueIfNull
    Else
        Nz = aValue
    End If
End Function
```

Here's an example of how to use the class module to pull the Customers data into a spreadsheet (without headers). We'll assume that the Customers table lives in an Access Database located here: `C:\MyDatabase.accdb`

```
Sub PullCustomersExcel()
    Dim cDB As New clsDB
    Dim rst As DAO.Recordset
    Dim sql As String

    sql = "SELECT * FROM Customers;"

    With cDB
        .Connection = "C:\MyDatabase.accdb"
        Set rst = .OpenRecordset(sql, dbOpenSnapshot)
    End With

    ThisWorkbook.Sheets(1).Range("A1").CopyFromRecordset rst

ExitLine:
    rst.Close
    Set rst = Nothing
End Sub
```

Here's an example of how use the class module to append a record to the Customers table and then print the newly created ID.

```
Sub AddCustomerRunSQLExcel()
    Dim cDB As New clsDB
    Dim sql As String
    Dim iID AS Long

    sql = "INSERT INTO Customers ( Name, State, Date, Sale ) " & _
            "SELECT 'Daniel Park' AS Name, 'CA' As State, #5/1/2019# As Date, 777.77␣
↪As Sale;"
```

(continues on next page)

(continued from previous page)

```
    With cDB
        .Connection = "C:\MyDatabase.accdb"
        .RunSQL sql
        iID = .DMax("ID", "Customers")
    End With

    Debug.Print iID
End Sub
```

2.3.11 excel - Concatenate Multiple

Excel does have a `CONCATENATE` function builtin, however it does not let you select a range, which can be particularly frustrating when trying to concatenate a handful of cells. Not to mention, if we would like to place delimiters in between the concatenation, we would have to type `& "/" &` between each cat. Here is a much more user friendly version of concatenation:

```
Function xx_cat(ref as Range, Optional ByVal delimiter as String = "") as String

    Dim cell as Range
    Dim result as String

    ' step through each cell and concatenate the results if the cell is not empty
    For Each cell in ref
        if IsEmpty(cell) = False Then
            result = result & cell.value & delimiter
        End If
    Next cell

    ' return the results without the last delimiter
    xx_cat = Left(result, len(result) - 1)

End Function
```

- Use it in a cell, where cell
  - A1=1
  - A2=2
  - A3=3
  - A4=4
  - A5=5

```
=xx_cat(A1:A5,"/")
>>> 1/2/3/4/5
```

2.3.12 excel - Cell Split

Surprisingly excel does not have a `split` function builtin. Ever run into a issue where you would like to pull out a part of a result based on common delimiters? Here is a function to solve just that.

- Function definition (this is placed in a module)

```
Function xx_cellsplit(value_to_split as String, _
                      Optional ByVal delimiter As String = "/", _
                      Optional ByVal Index as Integer = 1) As String:

    ' check if input value is empty and return empty to prevent error
    If value_to_split = "" Then
        xx_cellsplit = ""
    Else
        parts = split(value_to_split, delimiter)
        xx_cellsplit = parts(index)
    End If

End Function
```

- Use it in a cell

```
=xx_cellsplit("10/20/30/40/50","/",3)
>>> 30
```

2.3.13 excel - Buffer Data From .csv Without Opening

A common output format for many different programs is the Comma Separated Values (csv). CSVs are great and easy to work with, but in order to being working with the data in excel we first have to copy it in. This can be a very annoying manual task that is also prone to mis-selection error. The following piece of code fixes this problem by buffering the CSV file into excel without ever launching a window for the csv!

- Step 1: Enable Microsoft Scripting Runtime. In the VBA Editor (Alt+F11) -> Tools -> References -> checkbox: Microsoft Scripting Runtime.

This will allow the object `FileSystemObject` to read data from the csv without pulling up a window. Note that this is a setting for the excel file itself that has to be done 1 time.

- Step 2: Built the CSV to Excel buffer code. This can be in a macro. VBA Editor -> Insert -> Macro

```
' set excel attributes to manual for better performance
Application.ScreenUpdating = False
Application.Calculation = xlCalculationManual


' In case an csv filename was incorrect or doesnt exists in path
On Error GoTo Error_FileNotFound

' Declare the workbook we are working on
Dim WB as Workbook
Set SB = ThisWorkBook

' As for CSV filename. Note that this does not need to be an input box. See more␣
↪cleaver solution
'  with Worksheet_Change and a drop-down menu
Dim FileName as String
FileName = InputBox("Enter the full csv file name. (ex: data.csv)")

' Get current workbook file path (assuming csv data file is in the same folder)
Dim DirPath as String
DirPath = WB.Path
```

(continues on next page)

(continued from previous page)

```
' Full file path for the csv file
Dim FilePath as String
FilePath = dir_path & "/" & FileName

' Row/Column to start copying csv data to (this example "A1" or cell(1,1))
Dim StartRow as Integer
Dim StartCol as Integer
StartRow = 1
StartCol = 1

' Clear previous contents from cells (assuming we are buffering csv data to tab
↪"Sheet1")
Dim WS as WorkSheet
Set WS = WorkSheets("Sheet1")
WS.Cells.ClearContents


' Setup FileSystemObject that will buffer the csv file
Dim FSO as FileSystemObject
Set FSO = New FileSystemObject

with FSO
    With .OpenTextFile(FilePath, ForReading)
        if Not .AtEndOfStream Then .SkipLine

        Do Until .AtEndOfStream
            ' split file by its delimiter, in this case "," for csv
            LineItems = split(.ReadLine, ",")
            ' get the max columns for the current line of csv textline to iterate over
            Dim MaxCol as Integer
            MaxCol = Ubound(LineItems) - LBound(LineItems) + 1
            ' iterate over each item and buffer the csv data into each excel column␣
↪cell of the current row
            Dim col as Integer
            col = 0
            Do Until StartCol = MaxCol
                ' on csv index starts at 0, we want ours to start at the specified␣
↪StartCol
                WS.Cells(StartRow + row, StartCol + col).Value = LineItems(col)
                col = col + 1
            Loop
        row = row + 1
        Loop
    End With
End With

' reset excel attributes
Application.ScreenUpdating = True
Application.Calculation = xlCalculationAutomatic


Exit Sub

Error_FileNotFound:
    MsgBox("Error - Incorrect file name and/or path. Double check .csv filename and␣
↪path is correct")
    Exit Sub
```

(continues on next page)

(continued from previous page)

```
End Sub
```

2.3.14 office - Create Outlook Email

This macro can be used in any office application to generate an email in Outlook. Subject and HTMLBody are required arugments while the rest are optional. The `fSend` argument controls whether the email is automatically sent or displayed at the end (default is display). The `vAttachments` argument needs to be an array of filepaths. The `vEmbeddedImages` argument also needs to be an array of filepaths, but these also need to be referenced within the HTMLBody (will expand upon this later).

```
Public Sub Create_Email(ByVal sSubject As String, _
                        ByVal sHTML As String, _
                        Optional ByVal sTo As String = "", _
                        Optional ByVal sCC As String = "", _
                        Optional ByVal sBC As String = "", _
                        Optional ByVal sOnBehalfOf As String = "", _
                        Optional ByVal fSend As Boolean = False, _
                        Optional ByVal vAttachments As Variant, _
                        Optional ByVal vEmbeddedImages As Variant)
    Dim olApp As Object              'Outlook Application
    Dim olMail As Object             'Outlook MailItem
    Dim olRecipient As Object        'Outlook Recipient
    Dim olAttach As Object           'Outlook Attachment
    Dim oPropAcc As Object           'Attachment Property Accessor
    Dim iAttch As Integer            'Attachment Counter
    Dim sTempDir As String           'Temp Directory
    Dim oFSO As Object               'File System Object

    Const PR_ATTACH_MIME_TAG = "http://schemas.microsoft.com/mapi/proptag/0x370E001E"
    Const PR_ATTACH_CONTENT_ID = "http://schemas.microsoft.com/mapi/proptag/0x3712001E
↪"
    Const PR_ATTACHMENT_HIDDEN = "http://schemas.microsoft.com/mapi/proptag/0x7FFE000B
↪"

    'Create Outlook Objects
    Set olApp = CreateObject("Outlook.Application")
    Set olMail = olApp.CreateItem(0)

    'Create Email
    With olMail
        .To = sTo
        .CC = sCC
        .BCC = sBC
        .Subject = sSubject
        .HTMLBody = sHTML
        If Len(sOnBehalfOf) > 0 Then
            .SentOnBehalfOfName = sOnBehalfOf
        End If

        'Embed Images
        If Not IsMissing(vEmbeddedImages) Then
            For iAttch = 0 To UBound(vEmbeddedImages)
                Set olAttach = .Attachments.Add(vEmbeddedImages(iAttch))
                Set oPropAcc = olAttach.PropertyAccessor
```

(continues on next page)

(continued from previous page)

```
                    oPropAcc.SetProperty PR_ATTACH_MIME_TAG, "image/jpg"
                    oPropAcc.SetProperty PR_ATTACH_CONTENT_ID, "item" & iAttch + 1
                    oPropAcc.SetProperty PR_ATTACHMENT_HIDDEN, True
            Next
        End If

        'Remove Temp Folder
        Set oFSO = CreateObject("Scripting.FileSystemObject")
        sTempDir = Dir(Environ("Temp") & "\TempHTML*", vbDirectory)
            If Len(sTempDir) > 0 Then
                Do
                    oFSO.DeleteFolder Environ("Temp") & "\" & sTempDir
                    sTempDir = Dir
                Loop Until Len(sTempDir) = 0
            End If
        Set oFSO = Nothing

        'Add Attachments
        If Not IsMissing(vAttachments) Then
            For iAttch = 0 To UBound(vAttachments)
                .Attachments.Add vAttachments(iAttch)
                Kill vAttachments(iAttch)
            Next
        End If

        'Resolve Recipients
        For Each olRecipient In .Recipients
            olRecipient.Resolve
        Next

        'Send or Display
        If fSend Then
            .Send
        Else
            .Display
        End If
    End With

ExitLine:
    On Error GoTo 0
    Set olApp = Nothing
    Set olMail = Nothing
End Sub
```

2.3.15 office - Hidden Window Background Check

Many times, when automating Microsoft Office, you'll want to keep the application window hidden from the user. If your code breaks when this happens, the application will be left open in the background. Here is some code to 1) check if any application instances are open in the background and then 2) kill those instances.

```
Public Enum msAppType
    msAppAccess
    msAppExcel
    msAppWord
End Enum
```

(continues on next page)

(continued from previous page)

```
Public Function CountBackgroundProcess(ByVal msAppType As Integer) As Integer
    Dim oWin32, oList, oProcess As Object
    Dim vAppArr As Variant
    Dim iCount As Integer

    vAppArr = Array("MSACCESS", "EXCEL", "WINWORD")

    'Query Background Processes
    Set oWin32 = GetObject("winmgmts:{impersonationLevel=impersonate}!\\.\root\cimv2")
    Set oList = oWin32.ExecQuery("SELECT * FROM WIN32_Process " & _
                                 "WHERE Name='" & vAppArr(msAppType) & ".EXE' " & _
                                     "AND CommandLine Like '%-Embedding'")

    'Count
    iCount = oList.Count

ExitLine:
    CountBackgroundProcess = iCount
    Set oProcess = Nothing
    Set oList = Nothing
    Set oWin32 = Nothing
End Function
Public Function KillBackgroundProcess(ByVal msAppType As Integer) As Boolean
    Dim oWin32, oList, oProcess As Object
    Dim vAppArr As Variant
    Dim bErr As Boolean

    vAppArr = Array("MSACCESS", "EXCEL", "WINWORD")

    'Query Background Processes
    Set oWin32 = GetObject("winmgmts:{impersonationLevel=impersonate}!\\.\root\cimv2")
    Set oList = oWin32.ExecQuery("SELECT * FROM WIN32_Process " & _
                                 "WHERE Name='" & vAppArr(msAppType) & ".EXE' " & _
                                     "AND CommandLine Like '%-Embedding'")

    'Terminate
    If oList.Count > 0 Then
        For Each oProcess In oList
            If oProcess.Terminate <> 0 Then
                bErr = True
            End If
        Next
    End If

ExitLine:
    KillBackgroundProcess = Not bErr
    Set oProcess = Nothing
    Set oList = Nothing
    Set oWin32 = Nothing
End Function
```

2.4 Shell Guide

2.4.1 tool - vsCode

1) Download Java Developer Kit from: https://code.visualstudio.com/docs/java/java-tutorial
2) Run the installation. On windows 10 java will unpack to: `C:\Users\vkisf\AppData\Local\Programs\AdoptOpenJD 0.8.10-hotspot\bin`
3) Pull up VSCode and download the `Java Extension Pack`
4) Create a new project

4.1) Via Explorer, create a new folder and place a `.java` file in it for vsCode to regonize that this project is a java project

4.2) Via Command Palette (ctrl + shift + p) and look for `Java: Create Java Project`

5) To add external libraries, search for the `.jar` files from: jar-download.com

5.1) Add them to your project via: `Explorer > Java Projects > FolderName > Referenced Libraries` that's located in the bottom left corner. Hit the + sign next to Referenced Libraries and select the `.jar` files