ENGR 298: Engineering Analysis and Decision Making - Functions

Dr. Jason Forsyth
Department of Engineering
James Madison University

Functions

- A **function** is a block of code, independent of the main program, that performs some operation with its **arguments** and **returns** a result.

- When a function is **called**, the flow/control of the program is passed to the function. Upon completion/return, the program resumes from where it left off.

- A function that is attached to a class, is called a method (more on that next week)

A program is a set of instructions for **manipulating**, **moving/storing**, and **making decisions** on **data** in your computer.

Some Built-In Functions

- We have already used several in previous lectures.

print(**objects*, *sep*=' ', *end*='\n', *file*=*sys.stdout*, *flush*=*False*) ¶

Print *objects* to the text stream *file*, separated by *sep* and followed by *end*. *sep*, *end*, *file*, and *flush*, if present, must be given as keyword arguments.

max(*iterable*, ***[, *key*, *default*])

max(*arg1*, *arg2*, **args*[, *key*])

Return the largest item in an iterable or the largest of two or more arguments.

len(*s*)

Return the length (the number of items) of an object. The argument may be a sequence (such as a string, bytes, tuple, list, or range) or a collection (such as a dictionary, set, or frozen set).

```python
# make a list and use a built-in function to perform
# some operations
my_list = [3,-2,3,4,2,1]

# use min, max, and len built-in functions
max_value = max(my_list)
min_value = min(my_list)
length = len(my_list)

# use the print built-in function
print("Max is ",max_value,
      "Min is ", min_value,
      "Length is ",length)
```

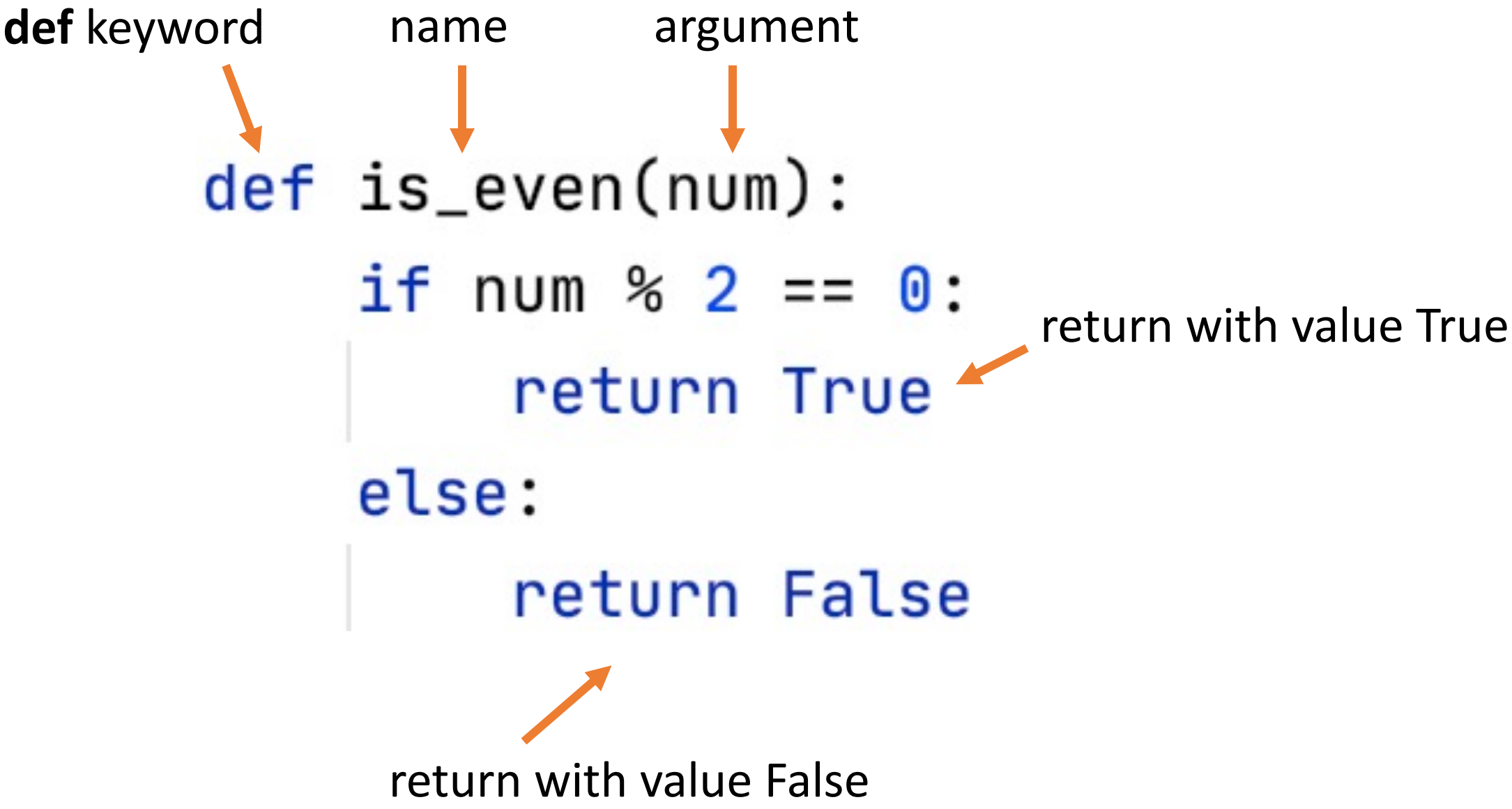
```
Max is  4 Min is  -2 Length is  6
```

Creating Functions in Python

- A function can be created by using the **def** keyword. A function *name* and then its *arguments* are provided.

- The arguments are only “in scope” within the function. Their values are not known outside. Should only operate on arguments Strongly avoid global variables.

- Functions can be defined “anywhere” but should generally be collected at the top of a file (C style), in another file (as module), or be associated with a class (method).


def keyword
name
argument
def is_even(num):
if num % 2 == 0:
return True
else:
return False
return with value True
return with value False

def keyword

name

argument

```
def average(nums):
    total = 0
    for n in nums:
        total += n

    return total / len(nums)
```

total is a *local variable* that only exists within average()

Use built-in function len()

Return with int/float result from calculation

Calling Functions

- To utilize a function, its name must be “known” to the program at the current context. This is called being **in scope**.

- Functions can be placed in scope by declaration in the same file or through import methods.

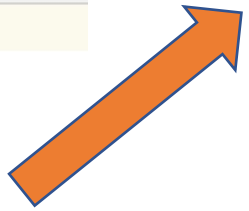
- Function names cannot/should not be the same as other variables, modules, or classes. Will lead to namespace conflicts.

```python
# make a helper function to calculate the average of some list
def average(nums):
    total = 0
    for n in nums:
        total += n

    return total / len(nums)


# create a new list of integers
new_list = [2, 3, 4, 5, 6]

# calculate the average
avg = average(new_list)
print("Average is: ",avg)
```

Importing Functions from Files and Modules

```python
import random


####
# Version 0.1 - JF
###

def generate_random_int_list(max_length, upper_bound):
    # generate random length between 2 and max_length
    list_length = int(random.uniform(2, max_length))

    # given the length above, sample the Natural Numbers u
    vars = random.sample(range(upper_bound), list_length)

    # return the generated list
    return vars
```

util.py



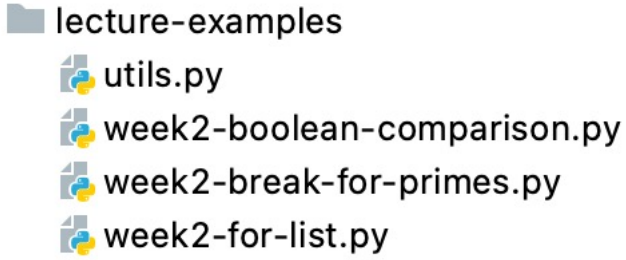
```python
from primality import primality
from utils import generate_random_int_list

# generate a long random list of integers
randoms = generate_random_int_list(1000, 10000000)

# iterate through the list check for all prime numbers
# if you see the magic number '100', gremlins are present and
# exit the loop. Keep track of how many we found
primes_found = 0
for p in randoms:

    # anything less than 2 is not prime
    if p < 2:
        continue

    # check to specific, dangerous numbers
    if p == 100:
        print("Danger!!")
        break
```

week2-break-for-primes.py

From *primality* package installed via pip

From local file ‘util.py’

Code runs but PyCharm is ☹ because utils is not a module. However, it finds the function as it’s in the same folder. Will dive into the differences between files, modules, and packages later.




```python
from primality import primality
from utils import generate_random_int_list

# generate a long random list of integers
randoms = generate_random_int_list(1000, 10000000)

# iterate through the list check for all prime numbers
# if you see the magic number '100', gremlins are present and
# exit the loop. Keep track of how many we found
primes_found = 0
for p in randoms:

    # anything less than 2 is not prime
    if p < 2:
        continue
```

week2-break-for-primes.py

Installing primality via PyCharm Terminal

```
(venv) jforsyth@Jasons-MacBook-Pro ENGR298-2022-Private % pip3 install primality
Collecting primality
  Downloading primality-0.0.6-py3-none-any.whl (3.9 kB)
Installing collected packages: primality
Successfully installed primality-0.0.6
```

Program Execution with Functions

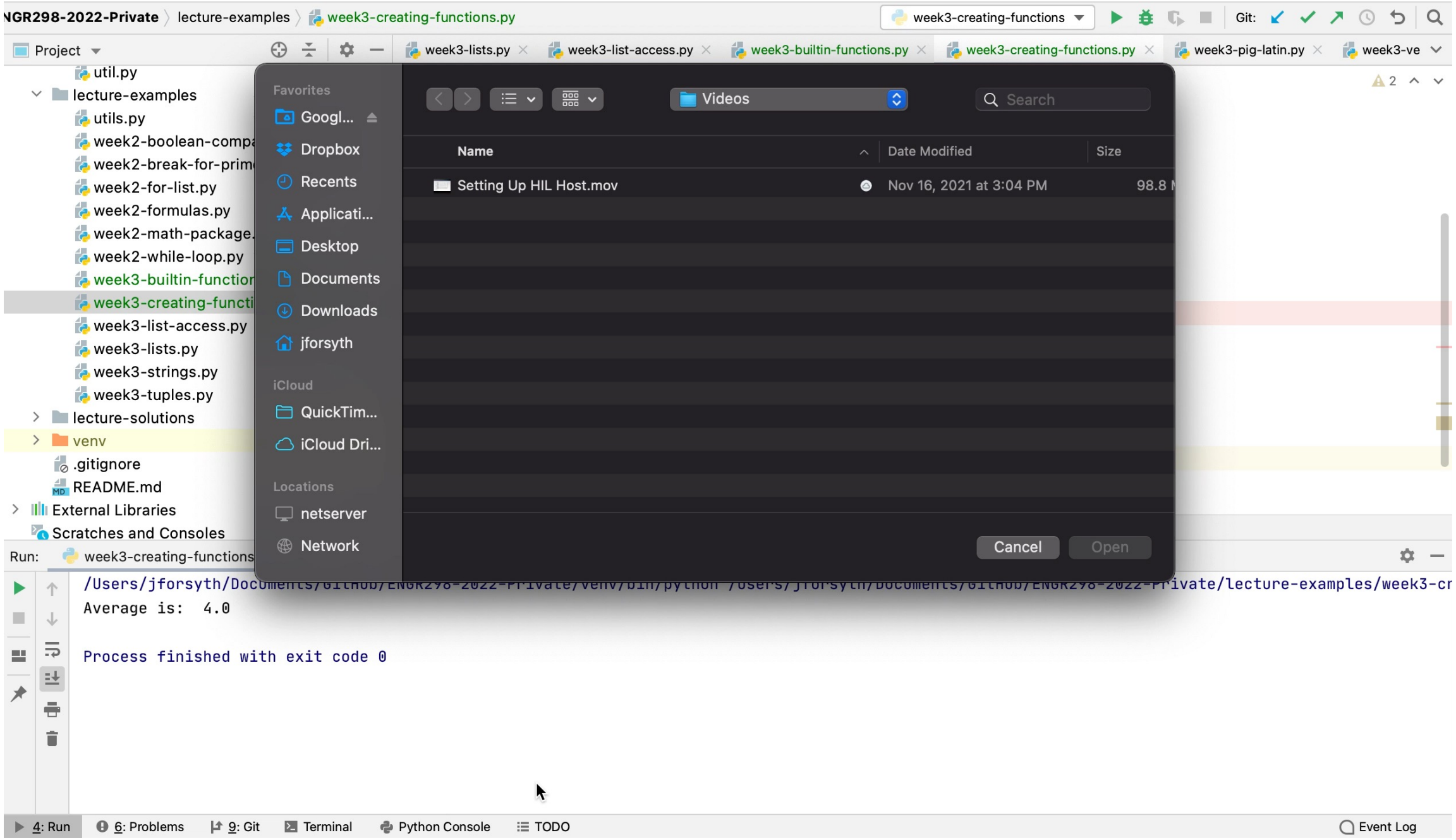
- A function is not executed unless it is **called**. In the previous example the program “begins” at the list declaration.

- When the function is “called”, the execution context switches to the function. While running the function, only variable passed to the function or created within are visible.

- Caveat: there are global, and static variables… but really avoid these.

```python
# make a helper function to calculate the average of some list
def average(nums):
    total = 0
    for n in nums:
        total += n

    return total / len(nums)


# create a new list of integers
new_list = [2, 3, 4, 5, 6]

# calculate the average
avg = average(new_list)
print("Average is: ",avg)
```

1
2
3
4
5
6


NGR298-2022-Private
lecture-examples
week3-creating-functions.py
week3-creating-functions
Git:
Project
week3-lists.py
week3-list-access.py
week3-builtin-functions.py
week3-creating-functions.py
week3-pig-latin.py
util.py
lecture-examples
utils.py
week2-for-list.py
week2-formulas.py
week2-while-loop.py
week3-list-access.py
week3-lists.py
week3-strings.py
week3-tuples.py
lecture-solutions
venv
.gitignore
README.md
External Libraries
Scratches and Consoles
Favorites
Dropbox
Recents
Desktop
Documents
Downloads
jforsyth
iCloud
Locations
netserver
Network
Videos
Search
Name
Date Modified
Size
Setting Up HIL Host.mov
Nov 16, 2021 at 3:04 PM
Cancel
Open
Run:
week3-creating-functions
Average is: 4.0
Process finished with exit code 0
4: Run
6: Problems
9: Git
Terminal
Python Console
TODO
Event Log

Candidates for Good Functions

- Code block is commonly used throughout program; creating function will save space, add clarity, and be easier to maintain.
- Function may be used in multiple contexts in various programs or other stages of the program (e.g. *generate_random_int_list*)
- Can operate solely on input arguments, does not need too many arguments, and return length is 1 - ~3 values.
- Function length would be short <30 lines. If result is longer, should be program into multiple functions or built out as larger Class.

Converting Code to Functions

1. Identify the input arguments. Should be of known/regular types. Include all data which function should make decision upon.

2. Identify the output/return types. Should be one or handful of values that meaning for result.

3. Write out the code without function. Test it. Once working, port to function.

4. Provide meaningful name to enhance readability.

What are the “inputs” to this function? What are their types?

$$A = P(1 + (r/100))^n$$

What are the “inputs” to this function? What are their types?

$$A = P(1 + (r/100))^n$$

Data	Variables Name	Units	Integer / Float
Principal	P	USD (could be any currency)	Two decimal point float
Interest rate	r	unit less	Floating point
Number of years	n	Years	Either; 1 year or 2.25 years

How many outputs? What is the type?

$$A = P(1 + (r/100))^n$$

How many outputs? What is the type?

$$A = P(1 + (r/100))^n$$

Data	Variables Name	Units	Integer / Float
Final amount	A	USD (could be any currency)	Two decimal point float

What is a good function name?

$$A = P(1 + (r/100))^n$$

```
def compound_return(P, r, years):
    final_value = P * (1 + (r / 100)) ** years

    return final_value


principal = 1000  # initial amount to be deposited
rate = 1  # interest rate applied to deposit (will be divided by 100)
n = 10  # number of years to compound deposit

# must cast n as 'string' so it can be printed in print()
print("Initial principal is $" + str(principal))
print("Interest rate is " + str(rate / 100))
print("Hold for " + str(n) + " years")

# calculate the eventual result
final = compound_return(principal, rate, n)

# fancy print the output with two decimal places for floating number
print(f"Final value after is ${final:.2f}")
```

This code runs and it REALLY REALLY shouldn't but Python lets you…

```
def compound_return(P, r, years):
    final_value = P * (1 + (rate / 100)) ** years

    return final_value


principal = 1000  # initial amount to be deposited
rate = 1  # interest rate applied to deposit (will be divided by 100)
n = 10  # number of years to compound deposit

# must cast n as 'string' so it can be printed in print()
print("Initial principal is $" + str(principal))
print("Interest rate is " + str(rate / 100))
print("Hold for " + str(n) + " years")

# calculate the eventual result
final = compound_return(principal, rate, n)
```

Returning multiple values via tuples

```
# find location (x,y) of minimum value in 2D array
def find_min_loc(array2D, x_dim, y_dim):

    # temp value to hold minimum value and its location
    min_value = array2D[0][0]
    x_loc = 0
    y_loc = 0

    # manually search through all (x,y) locations in the matrix
    # to find the minimum value and its location
    for idx in range(0,x_dim):
        for idy in range(0,y_dim):
            if array2D[idx][idy] < min_value:
                min_value = array2D[idx][idy]
                x_loc = idx
                y_loc = idy

    # return results as tuple
    return x_loc, y_loc, min_value
```

```
# get random length from 2 to 10
length = random.randint(2, 6)

# create square array of random length
array = np.random.rand(length, length)

# function returns multiple results
(x, y, value) = find_min_loc(array, length, length)

print("For array: ")
print(array)
print("Min value is", value, " located at ("+str(x)+","+str(y)+")")
```

```
For array:
[[0.0761755  0.02780509 0.6910647  0.83038148]
 [0.26291993 0.72451491 0.23803991 0.66037609]
 [0.9798341  0.83285007 0.03259465 0.8251261 ]
 [0.64112308 0.37592159 0.51532881 0.945979  ]]
Min value is 0.027805088417475 42  located at (0,1)
```

Since Python doesn’t have types, how do we know what ‘types’ to pass to a function or what it returns?

Welcome to documentation…

Code Documentation

- Almost as important as writing the code is documenting its logic, variables, and adding comments about decisions made.

- Future coders, employers, yourself… will need this information to maintain the program. Code that is unmaintainable will be used in limited settings and then trashed or re-written.

- Provide value for your future self. Document your code!

```
def _calc_sample_rate_and_start_time(self):
    # read the first 10 rows so we can get the sample rate
    #TODO: make this not hard coded. Will error is file is less than numRows
    numRows = 100
    data = pd.read_csv(self.filePath, nrows=numRows)
    times = data[self._epoch_column_label]
    deltas = times.diff()
```

```
for (start, end) in windows(data['timestamp'], window_size):
    feature_data = list()
    for feature in feature_list:
        val = data[feature][start:end]
        feature_data.append(val)

    #Data segment is of length WINDOW_SIZE. No padding/exceptions needed
    if (len(data['timestamp'][start:end]) == window_size):
        segments = np.vstack([segments, np.dstack(feature_data)])
        labels = np.append(labels, stats.mode(data["Activity"][start:end])[0][0])
    else:
        dummy=0
        #this is a place holder for the last data in the segment
        #anything in this condition is a segment of not WINDOW_SIZE length
        #possibly could add padding of zeros to make correct size. However
        #as currenlty written this segment is dropped
```

One-line Docstrings

One-liners are for really obvious cases. They should really fit on one line. For example:

```
def kos_root():
    """Return the pathname of the KOS root directory."""
    global _kos_root
    if _kos_root: return _kos_root
    ...
```

Notes:

- Triple quotes are used even though the string fits on one line. This makes it easy to later expand it.
- The closing quotes are on the same line as the opening quotes. This looks better for one-liners.
- There's no blank line either before or after the docstring.
- The docstring is a phrase ending in a period. It prescribes the function or method's effect as a command ("Do this", "Return that"), not as a description; e.g. don't write "Returns the pathname ...".
- The one-line docstring should NOT be a "signature" reiterating the function/method parameters (which can be obtained by introspection). Don't do:

If done correctly, will be automatically pulled by programming environment

```
# check to see if p is prime or not
primality_test = primality.isprime(p)

if primality_test == True:
    print("The value "+str(p)+
    primes_found = primes_foun

oop is now done. Print out the
nt("The list contained " + str(
```

primality.primality
def **isprime**(p: int) -> bool

True if p is prime.

Returns: True -- If {p} is prime.

< Python 3.8 (ENGR298-2022-Private)

>

```
def find_min_loc(array2D, x_dim, y_dim):
    """
    Find the (x,y) location and value for minimum in 2D array

    :param array2D: a two dimensional numpy array
    :param x_dim: length of x-dimension
    :param y_dim: length of y-dimension
    :return: (x location, y location, minimum value)
    """
    # temp value to hold minimum value and its location
    min_value = array2D[0][0]
    x_loc = 0
    y_loc = 0
```

Use PyCharm automated tools to generate the layout. Then fill in descriptions.

```
# function returns multiple results
(x, y, value) = find_min_loc(array, length, length)

print("For array: ")
print(array)
print("Min value is", va
```

```
/Users/jforsyth/Documents/GitHub/ENGR298-2022-Private/le
def find_min_loc(array2D: {__getitem__},
                 x_dim: Any,
                 y_dim: Any) -> Tuple[int, int, Any]
```

Find the (x,y) location and value for minimum in 2D array

Params: `array2D` – a two dimensional numpy array
`x_dim` – length of x-dimension
`y_dim` – length of y-dimension

Returns: (x location, y location, minimum value)

Weekly Assignments

- **Normal Distribution**: use the numpy method normal() to generate samples from an *ideal* normal distribution ($\mu = 0, \sigma = 1$). Then calculate μ and σ from the samples generated to see how close the sample distribution is to the ideal. Increase the number of samples until the error on μ and σ is <1E-3.

- **Functional PI**: implement a function called calculate_pi() that returns a value for π within a certain error bound. This error bound should be passed as a parameter to the function. For testing, set the initial error to 1E-10. Hint: breaking out of loops will be your friend. Hint 2: scientific notation in Python.

- **Python Doc Strings**: once the calculate_pi() program is complete. Add doc strings such that PyCharm (or whatever program you use) shows the function description, arguments, and return type as a pop-up in the GUI. See previous slide for good example. Upload screenshot to assignment.