



---

# Python Fundamentals, Data Structures, and Algorithms:

## Week 2 Workshop Presentation



# Workshop Agenda

Activity	Estimated Duration
Set up and check in	10 mins
Week 2 Review	60 mins
Assignment Tasks 1-3	50 mins
Break	15 mins
Assignment Tasks 4-5	100 mins
Check-Out (Feedback & Wrap-Up)	15 mins



# **Week 2 Review**



# Overview

For loops	Void functions
Using range()	Return values
Break & continue	Scope
Functions	Lambda functions
Built-in functions	Recursion
Type conversion	Modules & packages
Custom functions	Random



# Review: For loops

- Similar to while loops
- Loops must have an exit condition to prevent infinite loop
- For loops iterate a fixed number of times then exit
- To determine the fixed number of times, we can use the `range()` function or an iterable value such as a string or list



# Review: For loops

- The range() function uses start, stop, and step values
- You can also use an iterable value such as a string or list
  - In this case, the loop runs once per character or item
  - Iteration variable within each loop is equal to each char/item in sequence

```
print('Using Range')
for i in range(0,10,2):
    print(i)
print('Using a list')
for i in [0,2,4,6,8]:
    print(i)
```

```
PROBLEMS  OUTPUT  TERMINAL  DEBUG CONSOLE

Using Range
0
2
4
6
8
Using a list
0
2
4
6
8
```



# Review: Break and continue

- Discuss: What do the break and continue keywords do?



# Review: Break and continue

- Discuss: What do the break and continue keywords do?
- Answer:
  - **break** exits the loop immediately.
  - **continue** skips the rest of the code in the current loop iteration and fast forwards to the next loop iteration





# Review: Functions

- Reuse and organize code
- Built-in and custom functions
- Syntax to call (invoke) a function:
  - *function\_name(arguments)*
- Question: What if there are no arguments?



# Review: Functions

- If there are no arguments, use an empty argument list:
  - *function\_name()*



# Review: Built-in functions

## Built-in Functions

The Python interpreter has a number of functions and types built into it that are always available. They are listed here in alphabetical order.

Built-in Functions				
<code>abs()</code>	<code>delattr()</code>	<code>hash()</code>	<code>memoryview()</code>	<code>set()</code>
<code>all()</code>	<code>dict()</code>	<code>help()</code>	<code>min()</code>	<code>setattr()</code>
<code>any()</code>	<code>dir()</code>	<code>hex()</code>	<code>next()</code>	<code>slice()</code>
<code>ascii()</code>	<code>divmod()</code>	<code>id()</code>	<code>object()</code>	<code>sorted()</code>
<code>bin()</code>	<code>enumerate()</code>	<code>input()</code>	<code>oct()</code>	<code>staticmethod()</code>
<code>bool()</code>	<code>eval()</code>	<code>int()</code>	<code>open()</code>	<code>str()</code>
<code>breakpoint()</code>	<code>exec()</code>	<code>isinstance()</code>	<code>ord()</code>	<code>sum()</code>
<code>bytearray()</code>	<code>filter()</code>	<code>issubclass()</code>	<code>pow()</code>	<code>super()</code>
<code>bytes()</code>	<code>float()</code>	<code>iter()</code>	<code>print()</code>	<code>tuple()</code>
<code>callable()</code>	<code>format()</code>	<code>len()</code>	<code>property()</code>	<code>type()</code>
<code>chr()</code>	<code>frozenset()</code>	<code>list()</code>	<code>range()</code>	<code>vars()</code>
<code>classmethod()</code>	<code>getattr()</code>	<code>locals()</code>	<code>repr()</code>	<code>zip()</code>
<code>compile()</code>	<code>globals()</code>	<code>map()</code>	<code>reversed()</code>	<code>__import__()</code>
<code>complex()</code>	<code>hasattr()</code>	<code>max()</code>	<code>round()</code>	

<https://docs.python.org/3.9/library/functions.html>



# Review: The input() function

Prompt for information from the user

```
username = input("What is your name? ")  
print("Welcome", username)
```

```
What is your name? Bilbo  
Welcome Bilbo
```

- When input() is called, code execution waits until user presses ENTER.
- input() always returns user input as a string value.
- Use assignment statement to assign input() return value to variable, so we can access the user input and use it.
- The user CAN press Enter without entering any data, so it's usually needed to check that data was entered.

```
while(True):  
    username = input("What is your name? ")  
    if username:  
        break  
print("Welcome", username)
```



# Review: Type conversion using built-in functions

Remember: The input() function always returns a string value

```
88 age=input('How old are you? ')
89 print('Next year you will be',age+1)
```

PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE

```
How old are you? 4
Traceback (most recent call last):
  File "examples.py", line 89, in <module>
    print('Next year you will be',age+1)
TypeError: can only concatenate str (not "int") to str
```

```
88 age=input('How old are you? ')
89 numeric_age=int(age)
90 print('Next year you will be',numeric_age+1)
```

PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE

```
How old are you? 25
Next year you will be 26
```

Built-in Function	Description
<code>str(arg)</code>	Argument passed in will be returned as String
<code>int(arg)</code>	Argument passed in will be returned as Integer Argument must contain a number
<code>float(arg)</code>	Argument passed in will be returned as Float Argument must contain a number



# Review: Custom functions

- Use **def** keyword
- Followed by function name
- Followed by parameter list and colon
- Parameter list gives variable names to the arguments passed in
- Must indent code block

```
✓ def add(x, y):  
    | z = x + y  
    | print(z)
```

```
add(2, 3)
```



# Review: Custom functions

```
#Calculate the area of a few triangles
base1=3
height1=10
area1=.5*base1*height1

base2=6
height2=15
area2=.5*base2*height2

base3=27
height3=12.3
area3=.5*base3*height3

base4=145
height4=83.8
area4=.5*base4*height4

print("The areas are:")
print(area1,area2,area3,area4)
```

```
The areas are:
15.0 45.0 166.05 6075.5
```

VS

```
#function name AOT short for Area Of Triangle
def AOT(base,height):
    ...return .5*base*height

print("The areas are:")
print(AOT(3,10),AOT(6,15),AOT(27,12.3),AOT(145,83.8))
```

```
The areas are:
15.0 45.0 166.05 6075.5
```

**DRY = Don't  
Repeat Yourself**



# Review: Void & value-returning functions

## Void function

```
def washingmachine(stufftowash):  
    filltub()  
    wash()  
    spin()  
    rinse()  
    spin()  
    buzz() #cycle is complete
```

## Value-returning function

```
def sodamachine(flavor, money):  
    inventory={  
        'cola':10,  
        'sugarola':0,  
        'mistimist':3,  
        'dietcola':2  
    }  
    if money==.50 and inventory[flavor]>0:  
        return flavor  
    else:  
        print('OUT - Try again')  
        return None
```

If the function is using the **return** keyword with a value/expression following it, then it is a **value-returning function**





# Review: Scope

```
92 gvar='G'
93 print('hello from global')
94
95 def func_a():
96     avar='A'
97     print('hello from func_a')
98
99     def func_b():
100         bvar='B'
101         print('hello from func_b')
102
103     print('Printing from func_a')
104     print(gvar)
105     print(avar)
106     func_b()
107     print(bvar)
108
109 #calling func_a
110 func_a()
```

- **Scope** defines the location from where you can access variables and functions within your Python code
- Global scope – not created in a function
- Local scope – created in a function
- Can be multiple levels of scope (due to nested functions)
- Child scope can access parent scope (the scope it is created in)
- A parent scope can not access variables and functions declared in any child scopes
- Question: What is the output of this code?



# Review: Scope

```
92 gvar='G'
93 print('hello from global')
94
95 def func_a():
96     avar='A'
97     print('hello from func_a')
98
99     def func_b():
100         bvar='B'
101         print('hello from func_b')
102
103     print('Printing from func_a')
104     print(gvar)
105     print(avar)
106     func_b()
107     print(bvar)
108
109 #calling func_a
110 func_a()
```

PROBLEMS	1	OUTPUT	TERMINAL	DEBUG CONSOLE
hello from global hello from func_a Printing from func_a G A hello from func_b Traceback (most recent call last): File "examples.py", line 110, in <module> func_a() File "examples.py", line 107, in func_a print(bvar) NameError: name 'bvar' is not defined				

Another question:

What will happen if we try calling func\_b() from a new line 111?



# Review: Scope

```
def func_a():  
    avar = "A"  
    print("hello from func_a")  
  
    def func_b():  
        bvar = "B"  
        print("hello from func_b")  
  
    print("Printing from func_a")  
    print(gvar)  
    print(avar)  
    func_b()  
    print(bvar)
```

func\_b()

NameError: name 'func\_b' is not defined



# Review: Scope

Global variables are generally to be avoided.

Remember: If you need to modify a global variable, you must add the **global** keyword along with the variable name inside your local scope – this provides a safeguard against accidentally modifying a global variable when you meant to modify a local variable.

```
gvar = "G"

def func_a():
    global gvar
    gvar = "g"  # make lower case

func_a()
print(gvar)
```

Result of print(gvar): 



# Review: Lambda functions

- Lambda functions are also known as anonymous functions, since they don't have names

```
115 def domath(opt, val, f1, f2):
116     if opt == 'sq':
117         returnval = f1(val)
118     elif opt == 'sqrt':
119         returnval = f2(val)
120     else:
121         returnval = None
122     return returnval
123
124 print(domath('sq', 4, lambda num: num ** 2, lambda num: num ** .5))
125 print(domath('sqrt', 9, lambda num: num ** 2, lambda num: num ** .5))
126 print(domath('duh', 25, lambda num: num ** 2, lambda num: num ** .5))
```

```
16
3.0
None
```



```
128 def factorial(num):
129     if num==1:
130         return num
131     else:
132         return num*factorial(num-1)
133
134 #5*4*3*2*1 = 120
135 print(factorial(5))
136 #10*9*8*7*6*5*4*3*2*1 = 3628800
137 print(factorial(10))
138 print(factorial(100))
139 print(factorial(998))
140 print(factorial(999))
```

Discuss: What is the base condition (or “base case”) for this recursive function, and how do you know?



# Review: Recursion

```
128 def factorial(num):
129     if num==1:
130         return num
131     else:
132         return num*factorial(num-1)
133
134 #5*4*3*2*1 = 120
135 print(factorial(5))
136 #10*9*8*7*6*5*4*3*2*1 = 3628800
137 print(factorial(10))
138 print(factorial(100))
139 print(factorial(998))
140 print(factorial(999))
```

Answer: This is the base case because it causes the recursion to end



# Review: Modules & packages

A **module** is a set of related variables, functions and classes that are grouped together into a single .py file.

A **module** can access the variables, functions and classes of another module (.py file) by importing all or parts of the “other” module.

```
#coolmath.py
pi=3.14159

def AreaCircle(radius):
    return pi*radius**2
```

```
#mathhomework.py
from coolmath import *
problem1=AreaCircle(6.7)
problem2=AreaCircle(83)
print('PI',pi)
print('Answer 1',problem1)
print('Answer 2',problem2)
```

PROBLEMS    OUTPUT    TERMINAL    DEBUG CONSOLE

```
PI 3.14159
Answer 1 141.02597509999998
Answer 2 21642.41351
```





# Review: Modules & packages

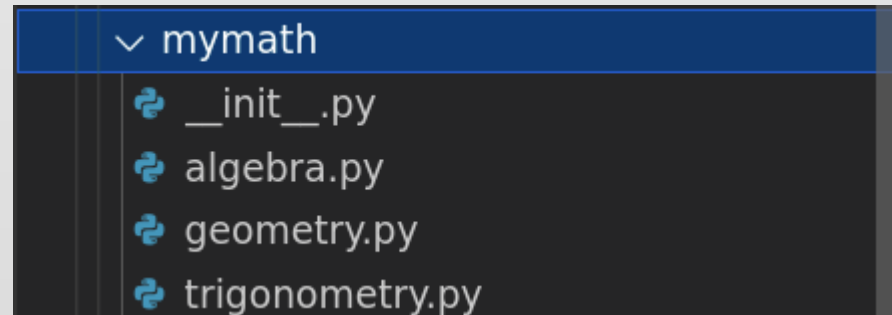
A **package** is a set of related modules grouped together into a folder.

A **package** should also contain an empty `__init__.py` file in the package folder along with the modules.

- **mymath** is a package
- **`__init__.py`** is required to denote a **regular** python package
- It contains 3 modules
  - `algebra.py`
  - `geometry.py`
  - `trigonometry.py`

Advanced Python:

Since python 3.3 you can create a **namespace package** that does not require the `__init__.py` file.





# Review: The random module

- Python built-in module
- Syntax:  
    `import random`
- Discuss: What does this code do?

```
print(random.randint(1, 100)) .....  
  
list = ['cherry', 'lemon', 'banana']  
print(random.choice(list)) .....  
  
random.shuffle(list) .....  
print(list) .....
```



# Review: The random module

```
print(random.randint(1, 100)) .....# prints random integer between 1-100 inclusive

list = ['cherry', 'lemon', 'banana'] .....
print(random.choice(list)) .....# prints random item from list

random.shuffle(list) .....# shuffles list item order
print(list) .....
```

Potential output:

```
66
cherry
['lemon', 'cherry', 'banana']
```



# Workshop 2 Assignment

Goal: Code an ATM application!

## **Task 1**

Set up the files and folders

## **Task 2**

Register a user

## **Task 3**

Log in the user and prompt to choose from the ATM menu

## **Task 4**

Create a banking package with useful banking-related functions

## **Task 5**

Import and use the banking package in your ATM app

You will be split up into groups to work on the assignment together.  
Talk through each step out loud with each other, code collaboratively.  
If your team spends more than 10 minutes trying to solve one problem,  
ask your instructor for help!