nucamp

Week 2 Workshop

Python Fundamentals, Data Structures, and Algorithms

Workshop Agenda

Activity	Estimated Duration
Welcome and check in	10 mins
Week 2 Review	75 mins
Break	15 mins
Workshop Assignment	2 hours
Code Review & Check-out	20 mins

Week 2 Review

Overview

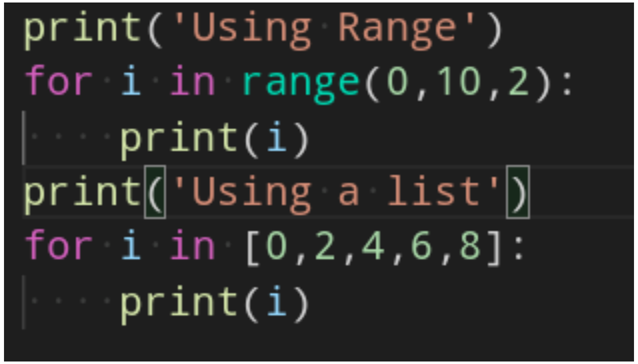
For loops	Void functions
Using range()	Return values
Break & continue	Scope
Functions	Lambda functions
Built-in functions	Recursion
Type conversion	Modules & packages
Custom functions	Random

Review: For loops

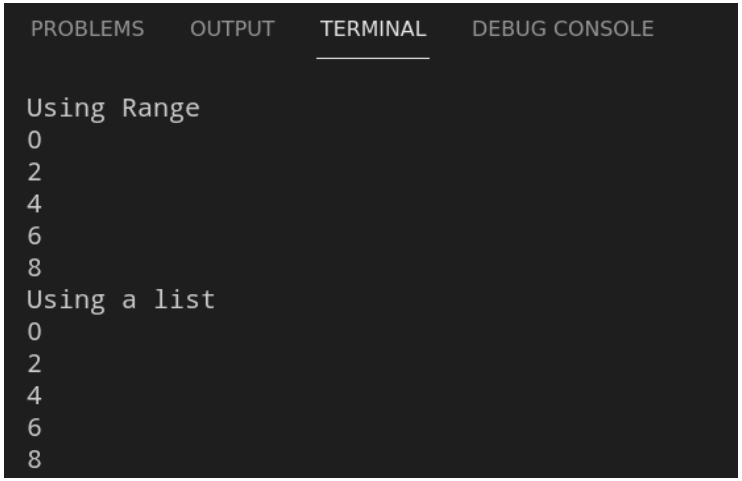
- Similar to **while** loops
- Loops must have an exit condition to prevent infinite loop
- For loops iterate a fixed number of times then exit
- To determine the fixed number of times, we can use the **range()** function or an iterable value such as a string or list

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)
```



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

Review: Break and continue

Discussion:

- What do the break and continue keywords do?

(answers on the next slide)

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)

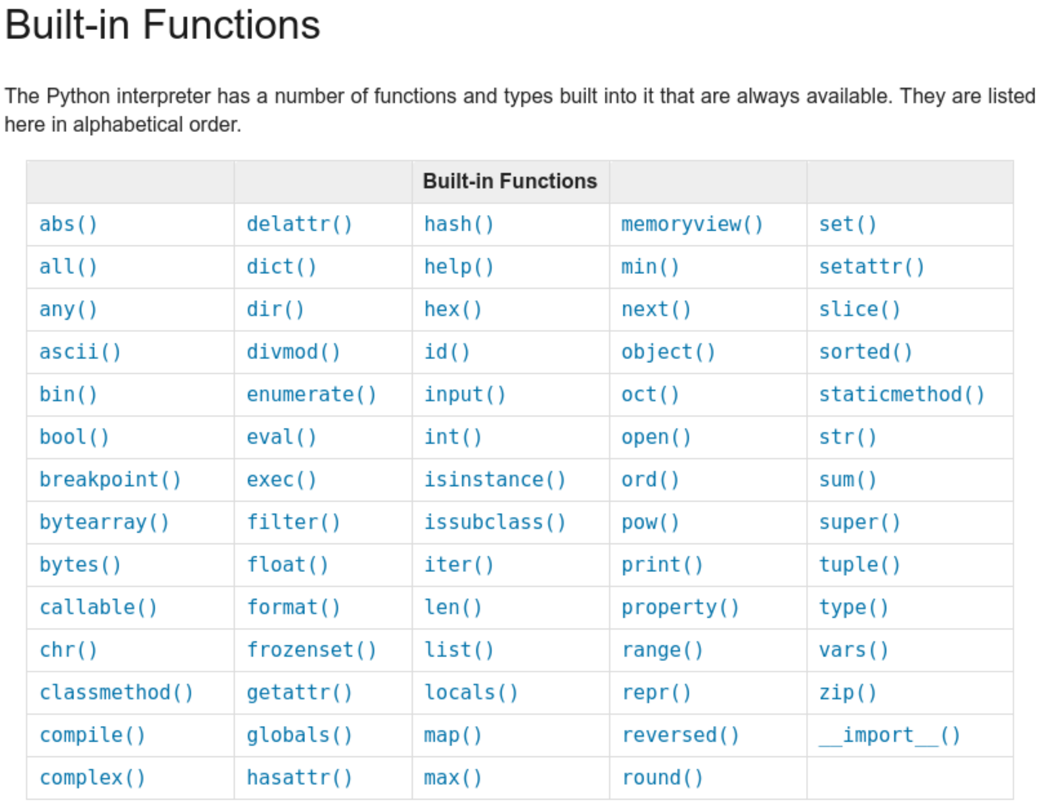
- Discussion: What if there are no arguments?

Review: Functions

- If there are no arguments, use an empty argument list:

  function_name()

Review: Built-in functions



Built-in Functions

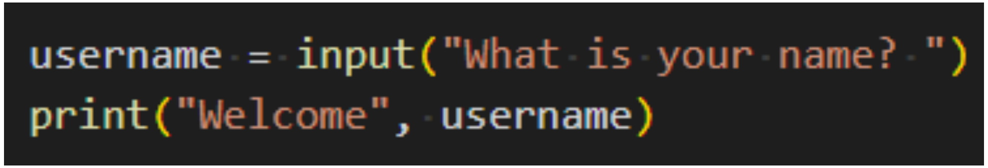
The Python interpreter has a number of functions and types built into it that are always available. They are listed here in alphabetical order.

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

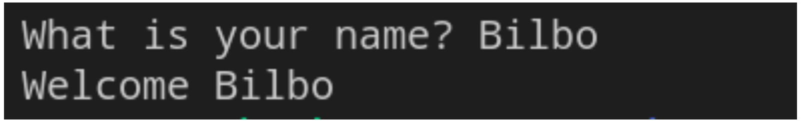
https://docs.python.org/3.9/library/functions.html

Review: The input() function

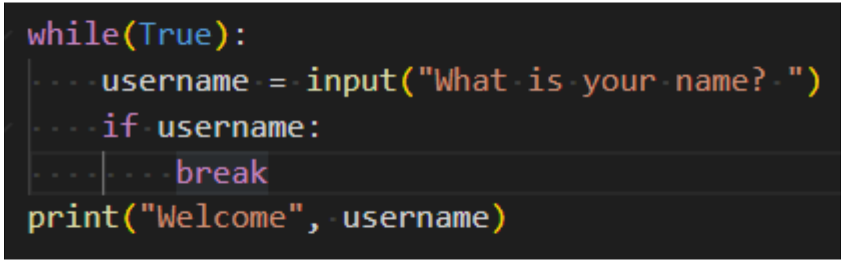
- Prompt for information from the user
- When input() is called, code execution waits until user presses ENTER.
- input() always returns user input as a string value.
- Use assignment statement to assign input() return value to variable, so we can access the user input and use it.
- You should check to make sure the user actually entered something and didn’t just hit ”enter”



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



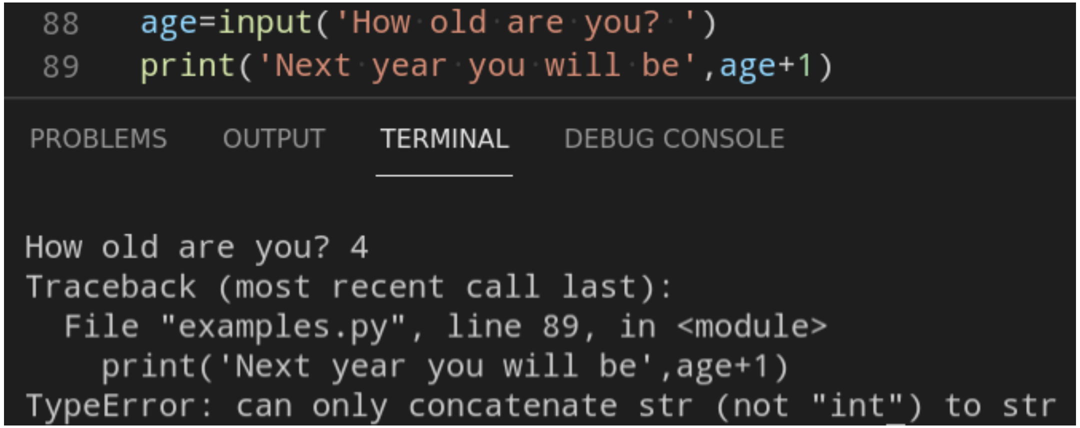
```
What is your name? Bilbo
Welcome Bilbo
```



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

Review: Type conversion using built-in functions

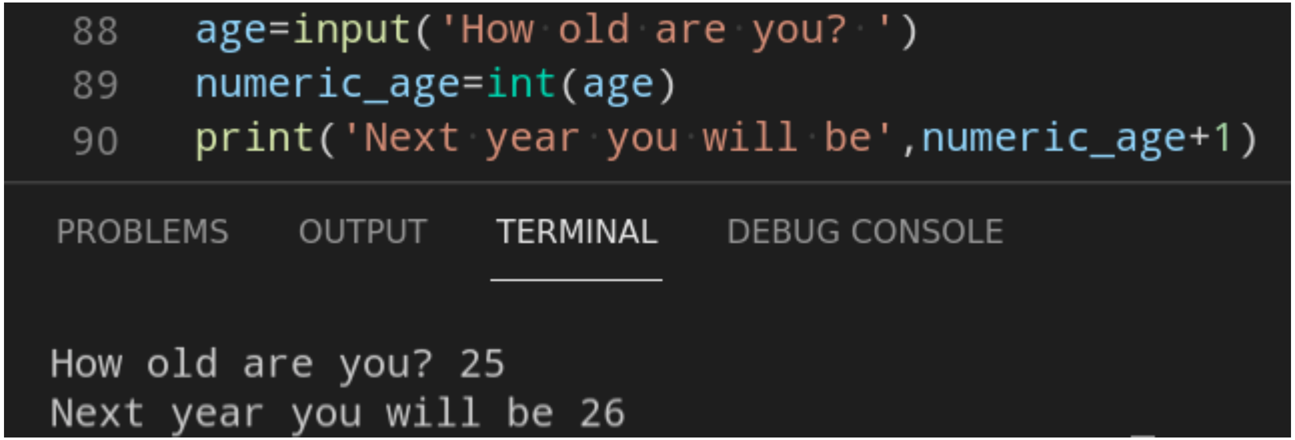
Remember: The input() function always returns a string value



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

```
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
```

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

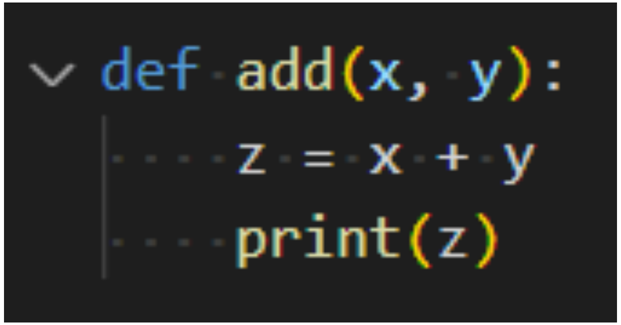


```
age=input('How old are you? ')
numeric_age=int(age)
print('Next year you will be',numeric_age+1)
```

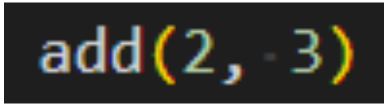
```
How old are you? 25
Next year you will be 26
```

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

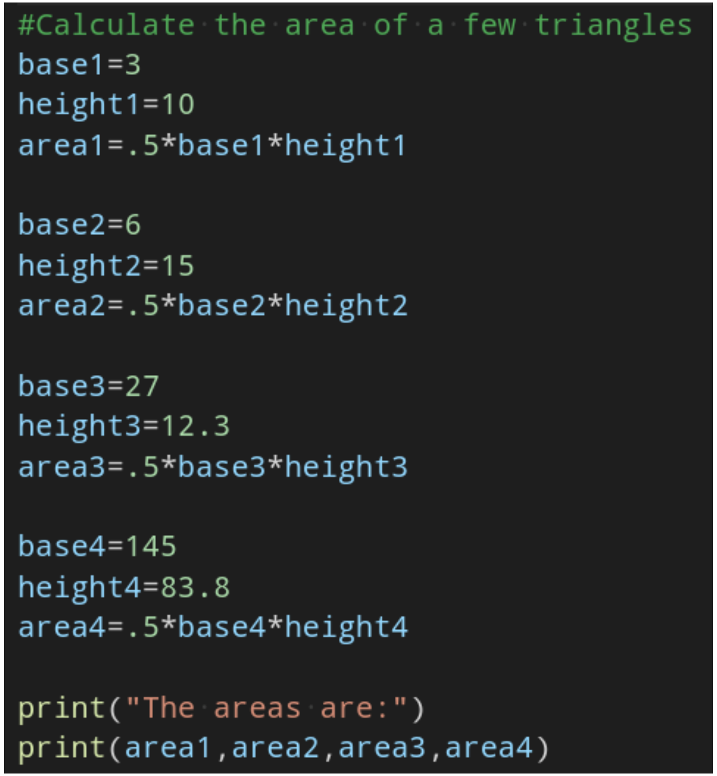


```python
def add(x, y):
    z = x + y
    print(z)
```

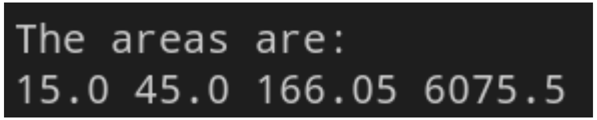


```python
add(2, 3)
```

Review: Custom functions

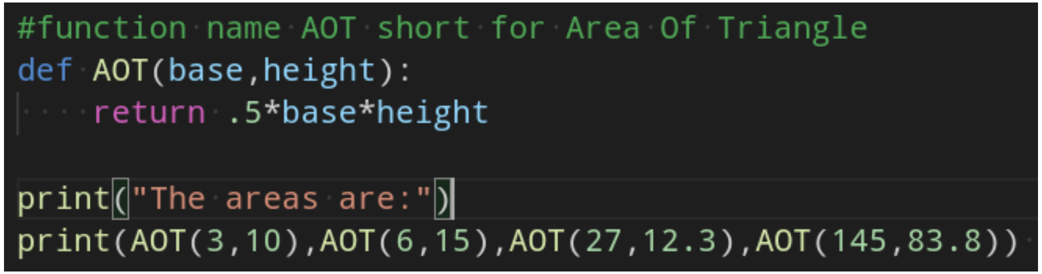


```
#Calculate the area of a few triangles
base1=3
height1=10
area1=.5*base1*height1

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)
```

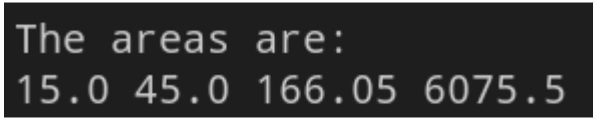


```
The areas are:
15.0 45.0 166.05 6075.5
```

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))
```

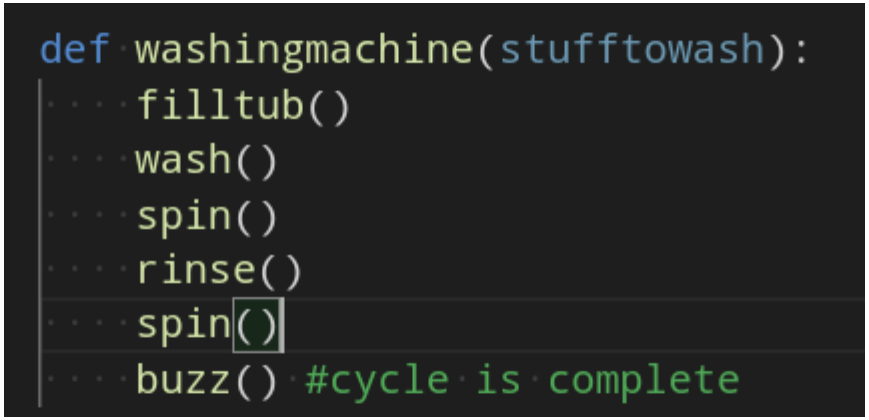


```
The areas are:
15.0 45.0 166.05 6075.5
```

DRY = Don’t Repeat Yourself

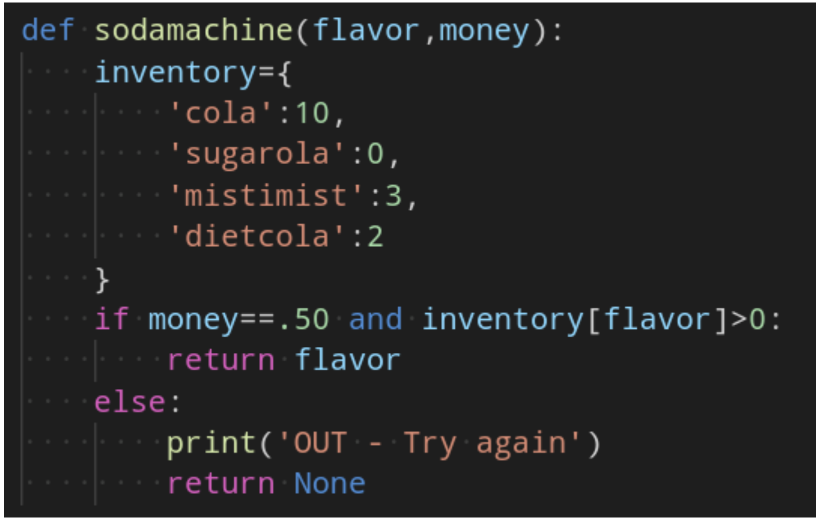
Review: Void & value-returning functions

Void function



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

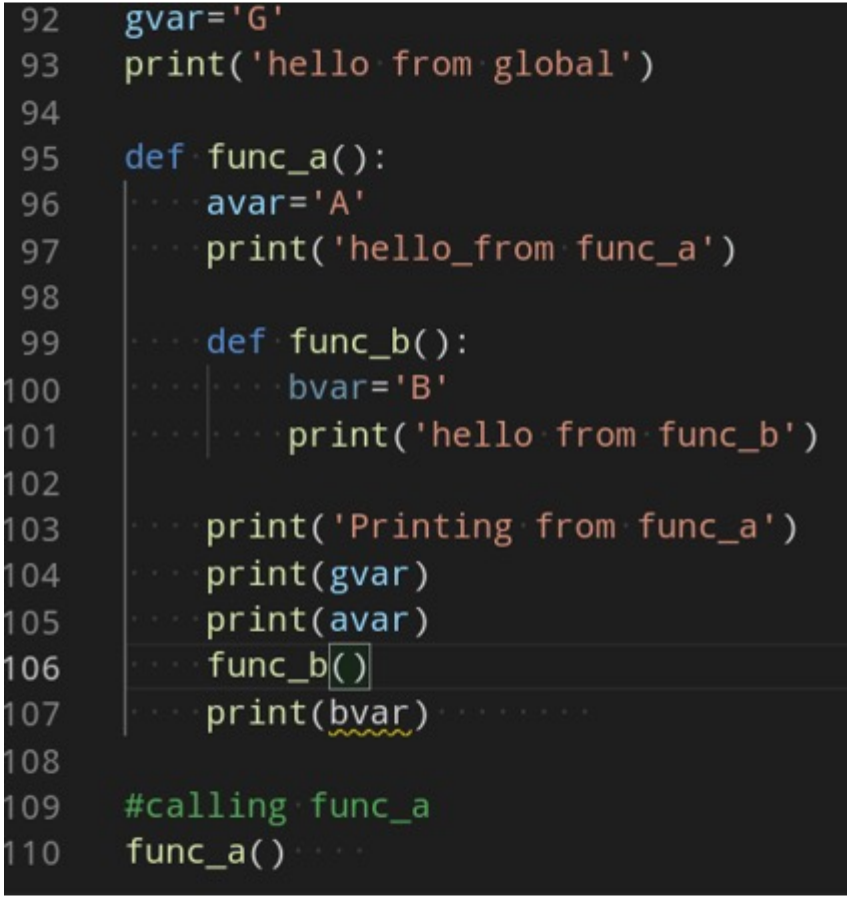
Value-returning function



```python
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**

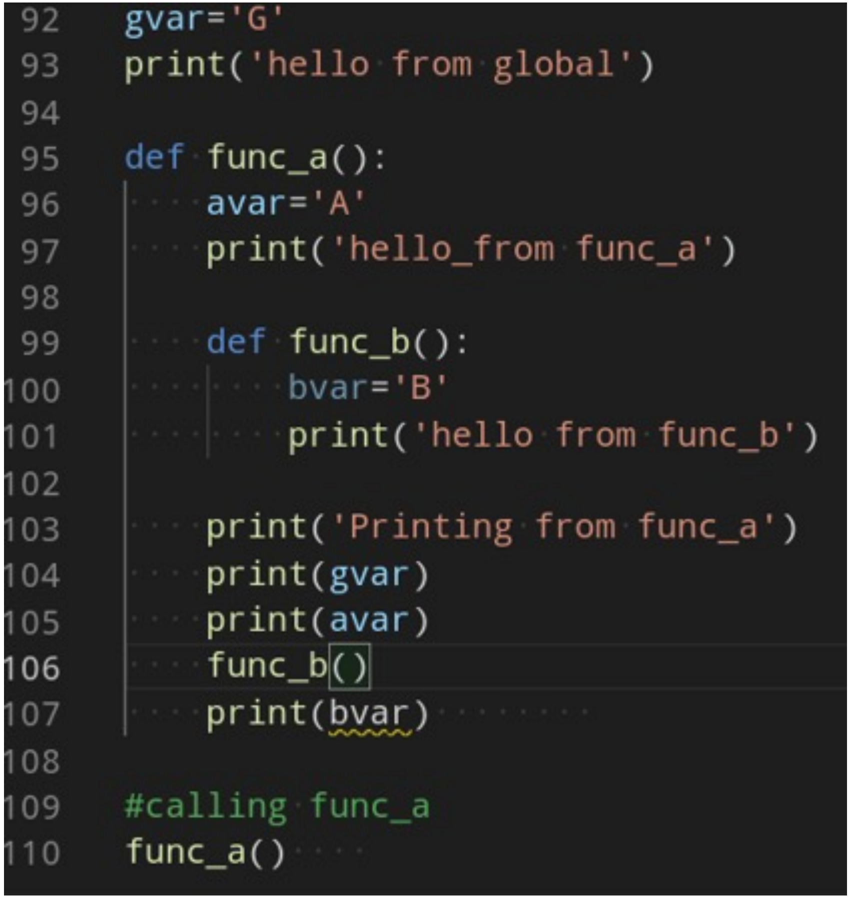
Review: Scope



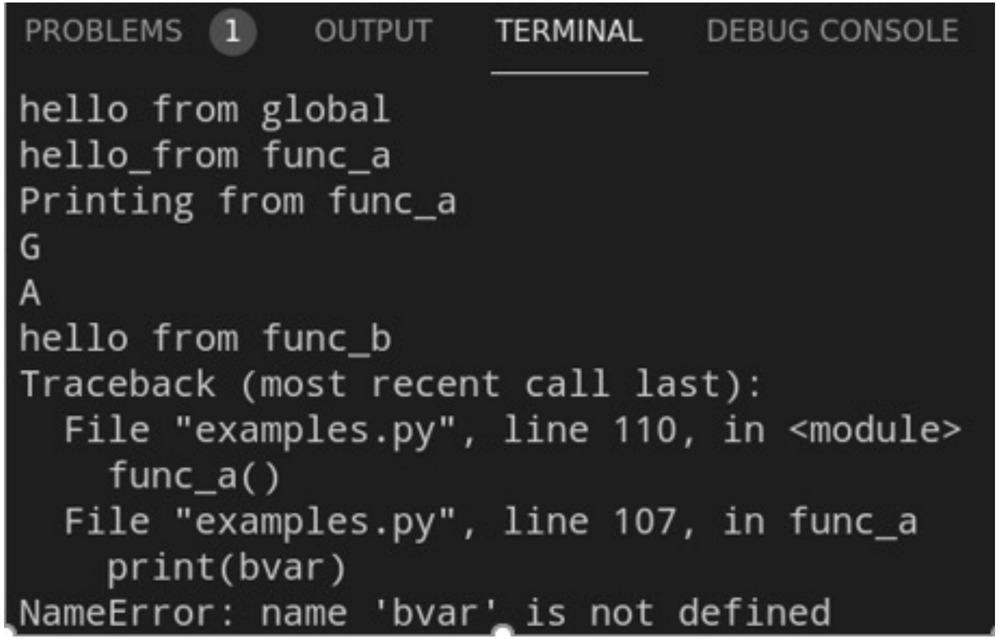
```python
gvar='G'
print('hello from global')

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)

#calling func_a
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



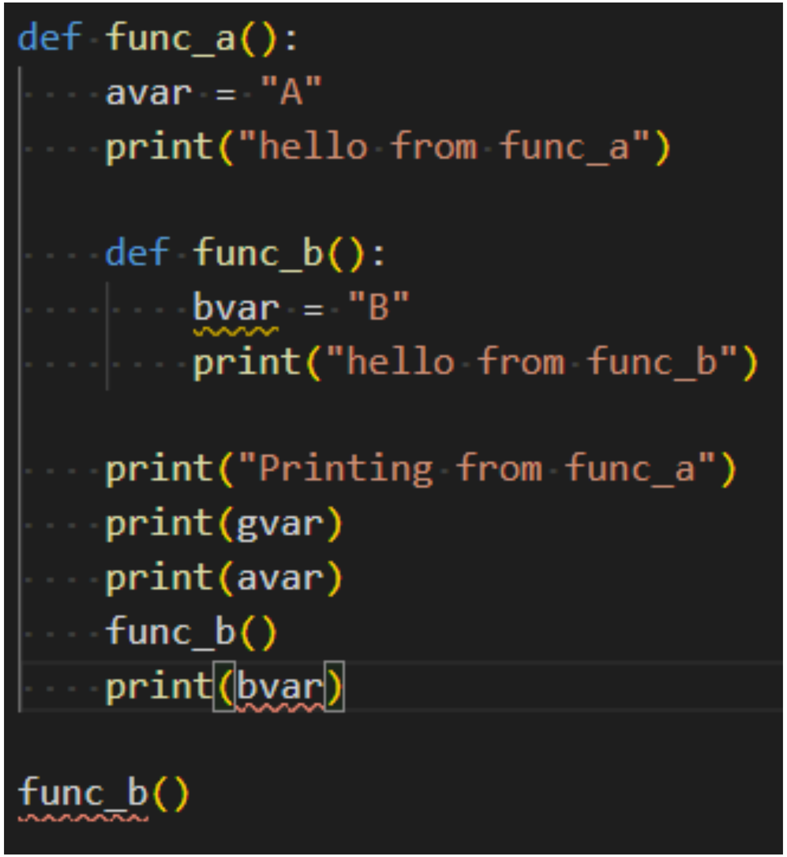
```
 92  gvar='G'
 93  print('hello from global')
 94
 95  def func_a():
 96      avar='A'
 97      print('hello_from func_a')
 98
 99      def func_b():
100          bvar='B'
101          print('hello from func_b')
102
103      print('Printing from func_a')
104      print(gvar)
105      print(avar)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



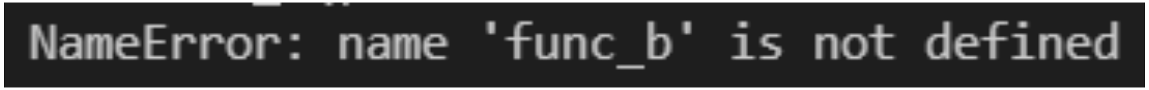
```
hello from global
hello_from func_a
Printing from func_a
G
A
hello from func_b
Traceback (most recent call last):
  File "examples.py", line 110, in <module>
    func_a()
  File "examples.py", line 107, in func_a
    print(bvar)
NameError: name 'bvar' is not defined
```

Another question:
What will happen if we try calling func_b() from a new line 111?

Review: Scope



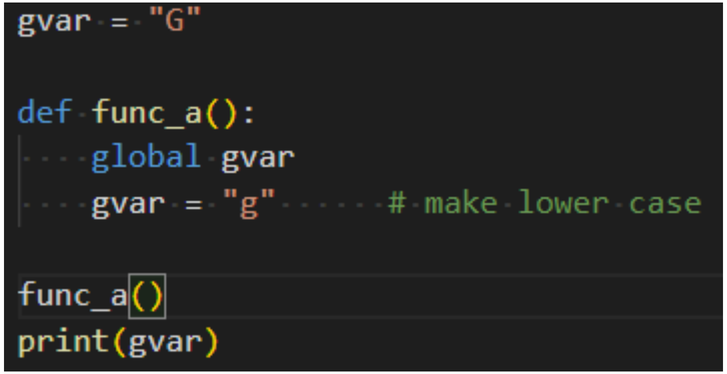
```python
def func_a():
    avar = "A"
    print("hello from func_a")

    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
```

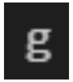
Review: Scope

- 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)
```

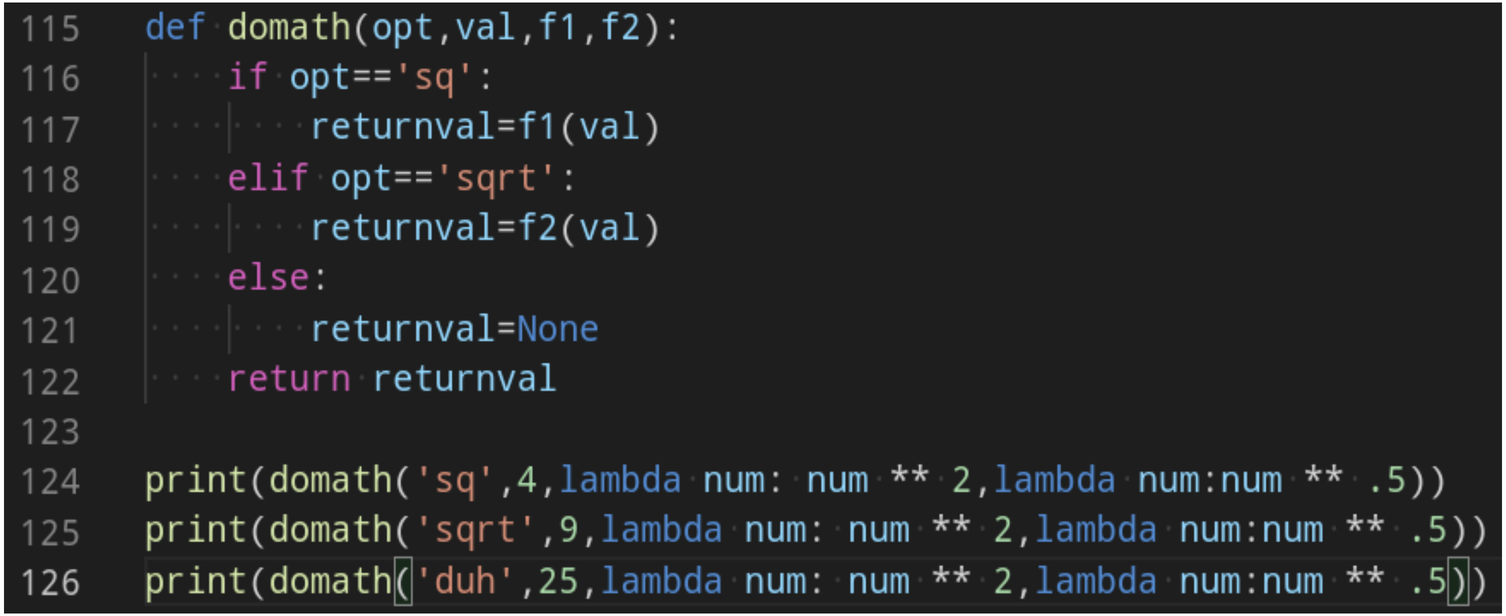
Result of print(gvar):



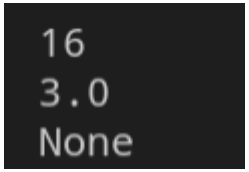
```
g
```

Review: Lambda functions

Lambda functions are also known as anonymous functions, since they don’t have names:



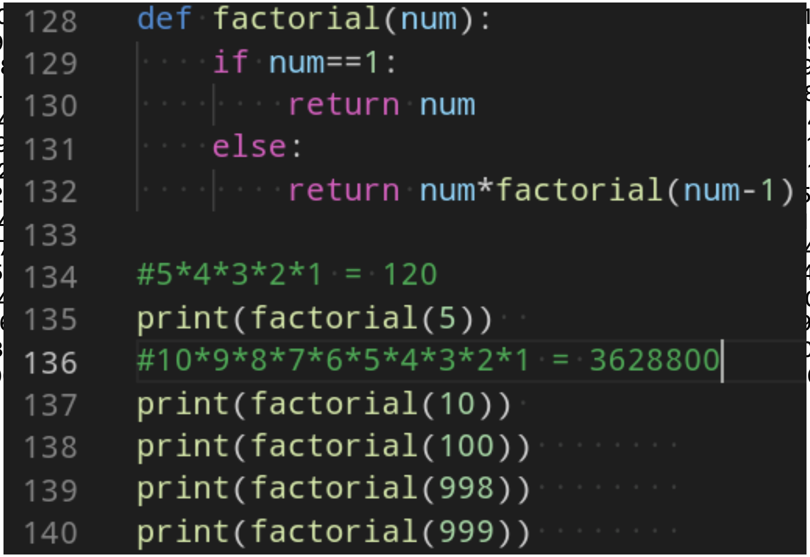
```python
def domath(opt,val,f1,f2):
    if opt=='sq':
        returnval=f1(val)
    elif opt=='sqrt':
        returnval=f2(val)
    else:
        returnval=None
    return returnval

print(domath('sq',4,lambda num: num ** 2,lambda num:num ** .5))
print(domath('sqrt',9,lambda num: num ** 2,lambda num:num ** .5))
print(domath('duh',25,lambda num: num ** 2,lambda num:num ** .5))
```



```
16
3.0
None
```

Review: Recursion

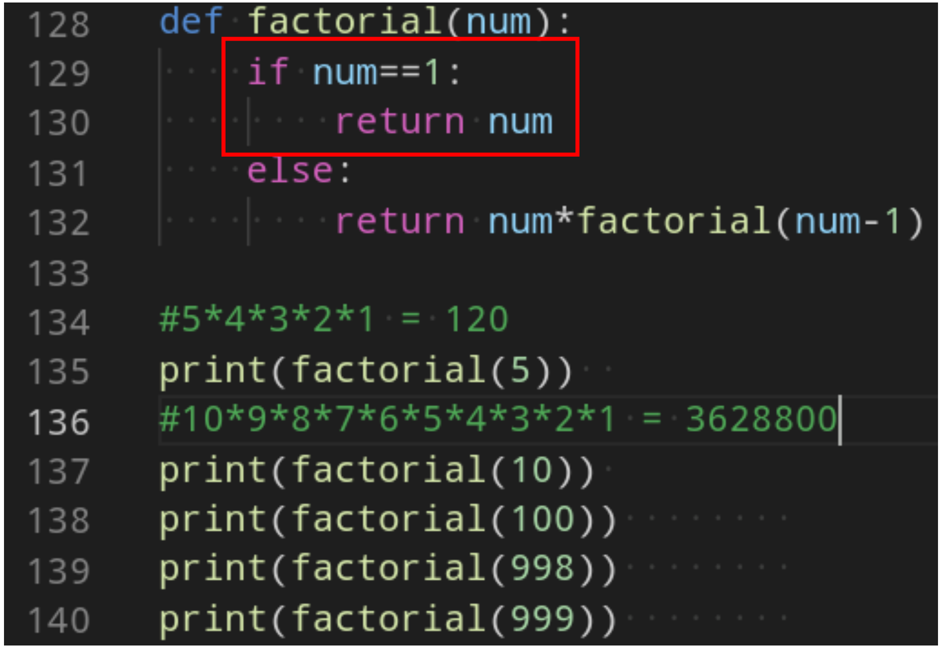
```
120
3628800
933262154439441526816992388562667004907159682643816214685929638952175999932299156089414639761565182862536979208272237582511852109168640000000000000000
00000000
402790050127220994538240674597601587306681545756471103647447357787726238637266286878923131618587992793273261872069265323955622495490298857759082912582
527118115540044131204964883707335062250983503282788739735011132006982444941985587005283378024520811868262149587473961298417598644470253901751728741217 85
07405765322677002133987226811442197771863 00                                 1450275256068755539376832805980594202740694146
568727386706899708796626357200339624064 39                                    934103235684110346477890399179387387649332 48351
085268065836314778365182198635137552922061 8                                  971779993325186354470006164529999840307397 15318
2191697073237996473757976873670132582033 64                                   898706207585962115186464083351842185711963964123
00835983314926628732700876798309217005024 4                                   414711426093563319610734142386307123138516605594
99144326959396112279901693382480279398435 9                                   130425983012915347763081242964010593797476 16677
85045203987508259776060285826091261745049 2                                   104372599880588166305491309198163384200635 4699
52551878482819585603303264547733812651266 2                                   577757503563031288598977986388832075922488212714
15443662515039749101007216506738103035770 74                                  4398263952929391318702517417558325636082722 9828
82372594816582486826728614633199726211273 07                                  487874996852498623584383106014533830650022 41105
36685081655478389620871112979473004444145 5                                   444138428582065142787356455528681114392680 95081
54182080723935326161223394344370344242878 4                                   009763612475872782742568849805927378373244 94619
07071684288078371462671562431852137243645 4                                   987595888189543123942343313277002244550158 717754
76100371615031940945098788894828812648426 3                                   000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000                                     000000000000000000000000000000000000000000000
000000000000
Traceback (most recent call last):
  File "examples.py", line 139, in <module>
    print(factorial(999))
  File "examples.py", line 133, in factorial
    return num*factorial(num-1)
  File "examples.py", line 133, in factorial
    return num*factorial(num-1)
  File "examples.py", line 133, in factorial
    return num*factorial(num-1)
  [Previous line repeated 995 more times]
  File "examples.py", line 130, in factorial
    if num==1:
RecursionError: maximum recursion depth exceeded in comparison
```



```
def factorial(num):
    if num==1:
        return num
    else:
        return num*factorial(num-1)

#5*4*3*2*1 = 120
print(factorial(5))
#10*9*8*7*6*5*4*3*2*1 = 3628800
print(factorial(10))
print(factorial(100))
print(factorial(998))
print(factorial(999))
```

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

Review: Recursion



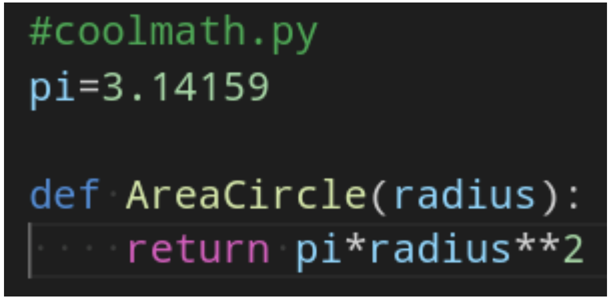
```
def factorial(num):
    if num==1:
        return num
    else:
        return num*factorial(num-1)

#5*4*3*2*1 = 120
print(factorial(5))
#10*9*8*7*6*5*4*3*2*1 = 3628800
print(factorial(10))
print(factorial(100))
print(factorial(998))
print(factorial(999))
```

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

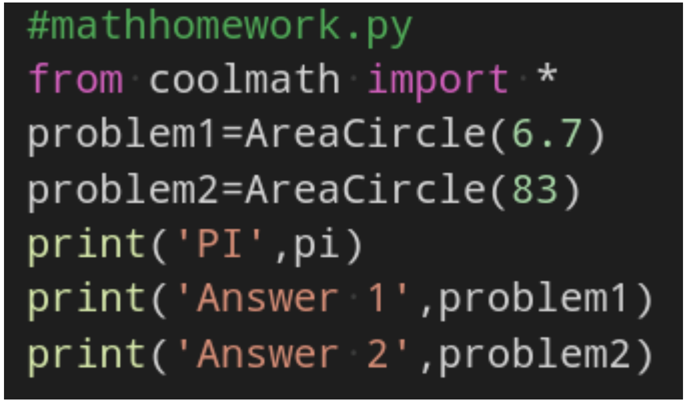
Review: Modules & packages

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

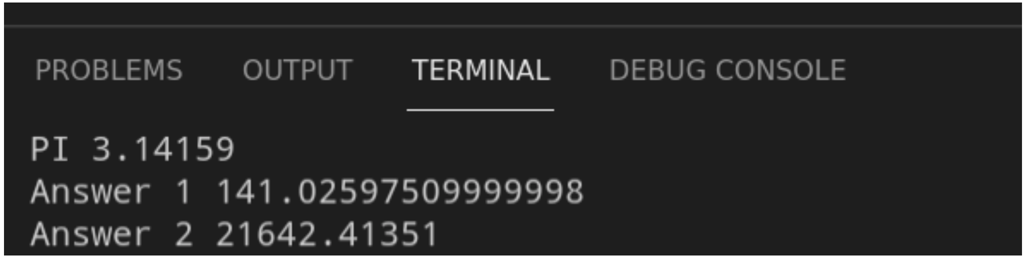
A **module** can access the variables, functions and classes of another module (.py file) by importing all or parts of the "other" module.



```python
#coolmath.py
pi=3.14159

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



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

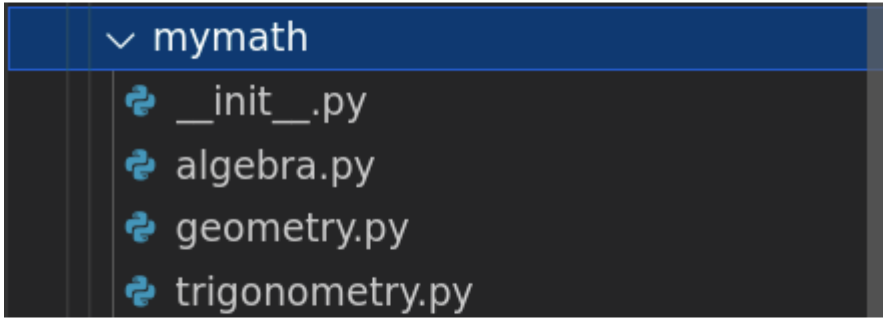


```
PI 3.14159
Answer 1 141.02597509999998
Answer 2 21642.41351
```

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

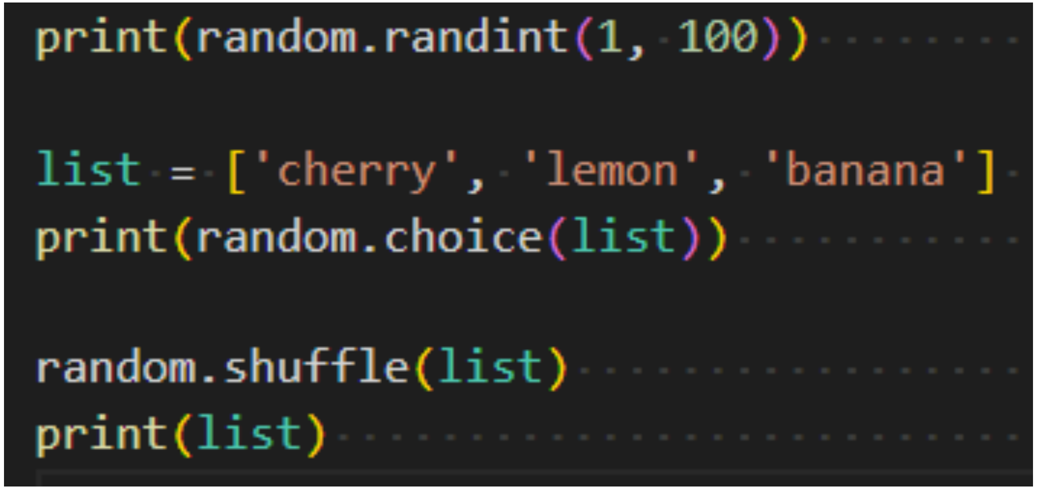
Advanced Python:
Since python 3.3 you can create a **namespace package** that does not require the __init__.py file.



mymath
- __init__.py
- algebra.py
- geometry.py
- trigonometry.py

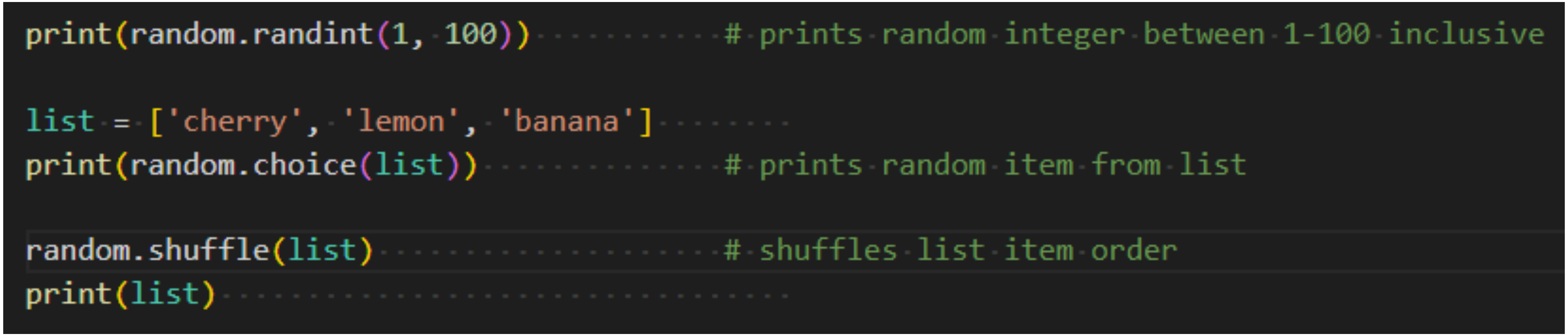
Review: The random module

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



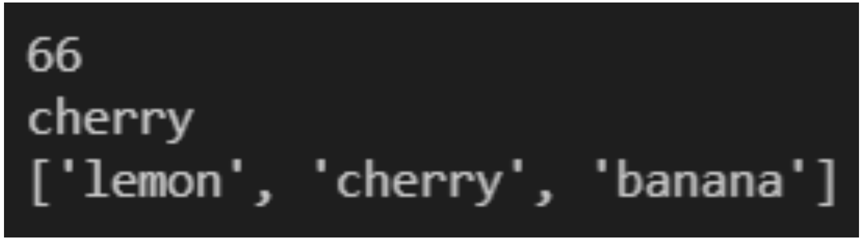
```
print(random.randint(1, 100))

list = ['cherry', 'lemon', 'banana']
print(random.choice(list))

random.shuffle(list)
print(list)
```

Review: The random module



```python
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!