COMP 2853

LESSON 2: FUNCTIONS AND MODULES

Agenda – Lesson 2

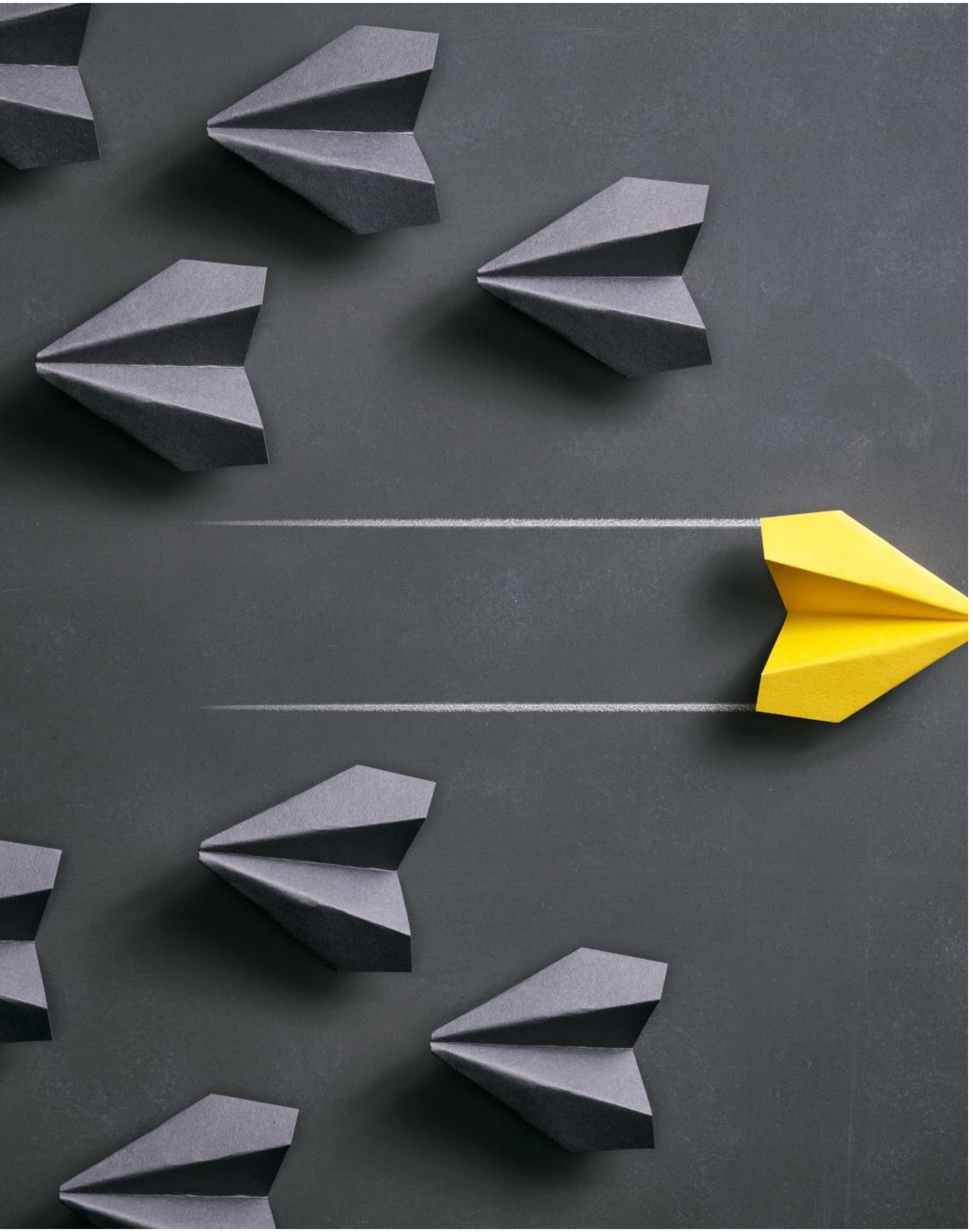
- Quick Review
  - Last Week’s Topics Lab
  - Pre-Reading
- Quiz 1
  - Review Answers
- Functions and Modules
- Lab 2 – In Class Part
- Homework – Lab 2 Take Home Part and Pre-Reading

Quiz 1

- This is an individual assessment, please do your own work

- You have 20 minutes to complete it

- We will go over the answers afterwards



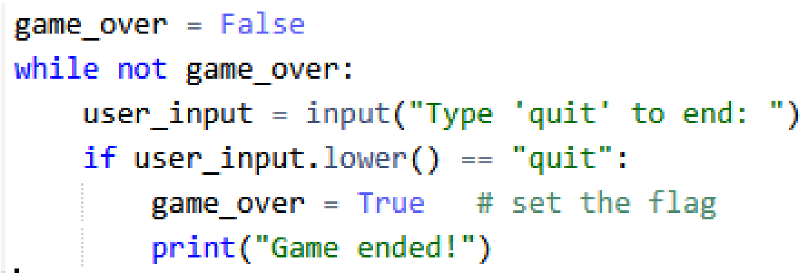
Lesson #2 Learning Outcomes

- Logical Operators
- Assignment Operators
- Functions
- Flow of Control
- Comments and DocString
- Modules
- Main Function

Boolean

- So far we have worked with int, float and str(string) data types.

- Boolean values are a fundamental data type to represent: True and False.
  - The uppercase first letters are important.
  - They are often used in conditional statements and logical operatons.
- Boolean variables can hold either True or False values (these are Boolean values, not Strings).



```
game_over = False
while not game_over:
    user_input = input("Type 'quit' to end: ")
    if user_input.lower() == "quit":
        game_over = True    # set the flag
        print("Game ended!")
```

- Boolean variables can be used as flags,
  - A flag is a variable that signals when some condition is met.

Boolean Expression

- Boolean expression is a comparison expression that would produce either True or False.
- Relational operators/ Logical operators can be used in Boolean expressions.

Relational Operators

- Relational operators are used to compare between values and evaluate to True or False.
- Relational operators can be used to compare between numbers or strings.

Relational Operator	Description	Example (Assume a = 3 , b = 4)
<	a < b means a is less than b	a < b ➔ True a < 2 ➔ False
>	a > b means a is greater than b	b > a ➔ True a > b ➔ False
<=	a <= b means a is less than or equal to b	a <= b ➔ True a <= 3 ➔ True a <= 2 ➔ False
>=	a >= b means a is greater than or equal to b	b >= a ➔ True b >= 4 ➔ True b >= 5 ➔ False

Equality Operators

- Equality Operators are a type of Relational Operator that compares if two values are equal or not equal.

Relational Operator	**Description**	**Example** **(Assume a = 3 , b = 4)**
==	a == b means a is equal to b	a == 3 ➔ True a == b ➔ False
!=	a != b means a is not equal to b	a != b ➔ True a != 3 ➔False

Equality Operators

- Note that = and == are different symbols
- = is an assignment operator; assign the value on the right to the variable on the left
- == is equality comparison operator that evaluates to True or False
  - number = 5 → assigns the value 5 to the variable number
  - number == 5 → True or False; does the variable number have the value 5?

Logical Operators

- In Python, logical operators are used to perform logical operations on Boolean values.
- Logical operators are used to combine or invert Boolean expressions. They operate on Boolean values and produce Boolean results.

Operator	Operator Syntax	Example
NOT equal	c!=d	4 not 5 is True; 4 not 4 is False not False is True ; ! True is False
and (vs. &)	c and d	True and True is True ; False and False is False False and True is False; True and False is False
or (vs. |)	c or d	False or True is True; False or False is False

& vs. &&
| vs. ||

Assignment Operators (Expanded)

compound assignment operator

Operator	Description	Example
= Assign	Assigns the value from right side operands to the left side operand	c = a + b
+= Add and	Adds right operand to the left operand and assigns the result to the left operand	c+= a is equivalent to c= c + a
-= Subtract and	It subtracts right operand from the left operand and assign the result to left operand	c -= a is equivalent to c = c - a
Do not use the operators below here: They are very confusing. Our main goal is to write clear code.		
*= Multiply and	It multiplies right operand with the left operand and assign the result to left operand	c *= a is equivalent to c = c * a
/= Divide and	It divides left operand with the right operand and assign the result to left operand	c /= a is equivalent to c = c / a
%= Modulus and	It takes modulus using two operands and assign the result to left operand	c %= a is equivalent to c = c % a
**= Exponent and	Performs exponential (power) calculation on operators and assign value to the left operand	c **= a is equivalent to c = c ** a
//= Floor division and	It performs floor division on operators and assign value to the left operand	c //= a is equivalent to c = c // a

Assignment Operators (Examples)

```
x = 6
x += 5  # x is now 11
x -= 2  # x is now 9
```

Note:

```
import math
math.floor(a/b)
```

```
x //= 4  # you just lost all your friends; do not do this

x = x // 4 # instead, do this
```

Note: import math math.ceil(x/b) or (a+b-1)//b

If you're writing a novel, your job is to be clear. Do not make your readers check a thesaurus or dictionary for every word.

Be simple. Be clear.

Operators Precedence

- By default, the descending order of the arithmetic operators in python is as follows:

Operator	Meaning
**	Exponent
* , /, // , %	Multiplication, Division, Floor Division, modulus
+ , -	Addition, Subtraction

- () Parentheses can be used to change the order of precedence of the operations

Operators Precedence

- Example:
    - 10 – 4 * 2 = 2 because the multiplication will be executed before the subtraction

      4 * 2 = 8   and then   10 – 8 = 2
    - Make it clearer! **10 - (4 * 2) Add in parentheses for clarity**
    - (10 – 4) * 2 = 12 with the parentheses the subtraction will be executed before the multiplication

      10 – 4 = 6 and then 6 * 2 = 12
- With the Parentheses the order of operation is:

Operator	Meaning
()	Parentheses
**	Exponent
* , / , // , %	Multiplication, Division, Floor Division, modulus
+ , -	Addition, Subtraction

Functions

- The function is a block of reusable code (Note: a group of related statements) that performs a specific task.
- Functions can be called procedures, sub-routines, or methods.
- **Functions allows us to divide the code into smaller modules which makes it easier to read, test, and use.**
- Functions can be either built in (e.g. print) or user defined (e.g. a function to solve a homework problem).

- Functions are more general and can return values.
- Procedures focus on performing tasks without returning values.
- Sub-routines are similar to functions and procedures but are often used in a more general context.
- Methods are functions that belong to objects and can operate on their internal state.

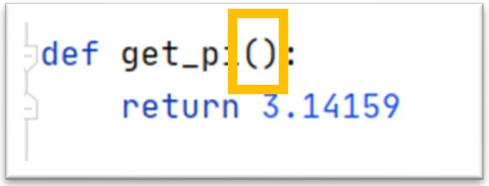
Functions

- Code Reusability
  - Functions allow you to write code once and reuse it multiple times without repeating yourself. This reduces redundancy and makes your code cleaner.
- Modularity
  - Functions help break down complex problems into smaller, manageable pieces. Each function can handle a specific task, making your code easier to understand and maintain.
- Improved Readability
  - Well-named functions provide context for what the code is doing, making it easier for others (or yourself) to read and understand later.
- Easier Testing and Debugging
- Maintainability
- Encapsulation
  - It can hide implementation details from the rest of the code. This helps prevent unintended interference with other parts of your program.

Function Parameters

- Functions may or may not receive parameters a.k.a. arguments.
- Example:

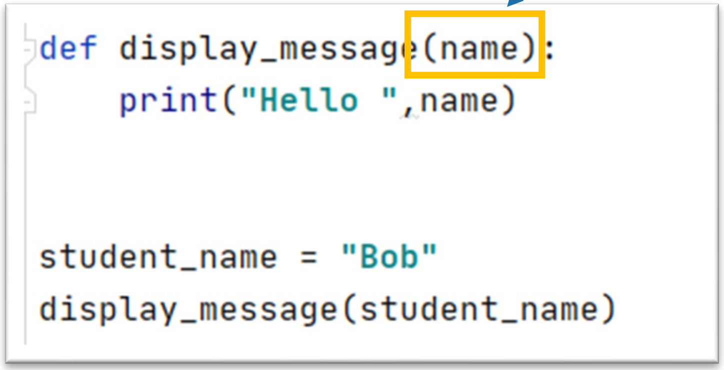
Function that does not receive any parameter(s)



```
def get_pi():
    return 3.14159
```

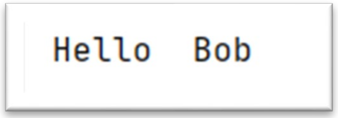
print(get_pi()) # empty ()

Function that receive parameter(s)



```
def display_message(name):
    print("Hello ",name)


student_name = "Bob"
display_message(student_name)
```

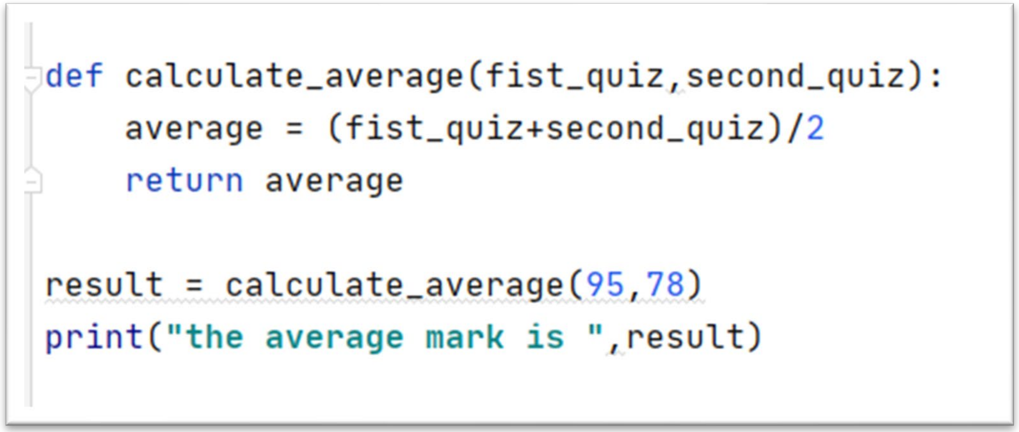
Output:



```
Hello  Bob
```

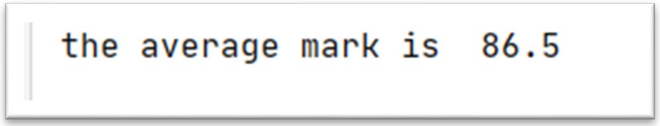
Function Parameters

- Functions may take more than one parameter that are separated by commas.
- If no parameters are given, then the function should be defined with an empty set of parenthesis
- Example:



```
def calculate_average(fist_quiz,second_quiz):
    average = (fist_quiz+second_quiz)/2
    return average

result = calculate_average(95,78)
print("the average mark is ",result)
```

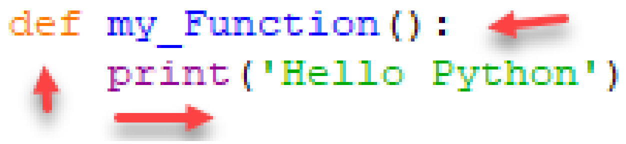
- Output



```
the average mark is  86.5
```

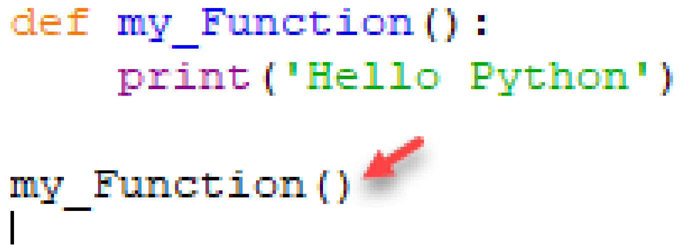
Creating and Calling a Function

- Define a function
  - It starts with a def statement that consists of the function name, a set of parentheses, an optional list of arguments with the parentheses, and a colon.
  - We indent the body of the function (exp. print()→)
  - The name of the function should be a verb that describes the action it performs



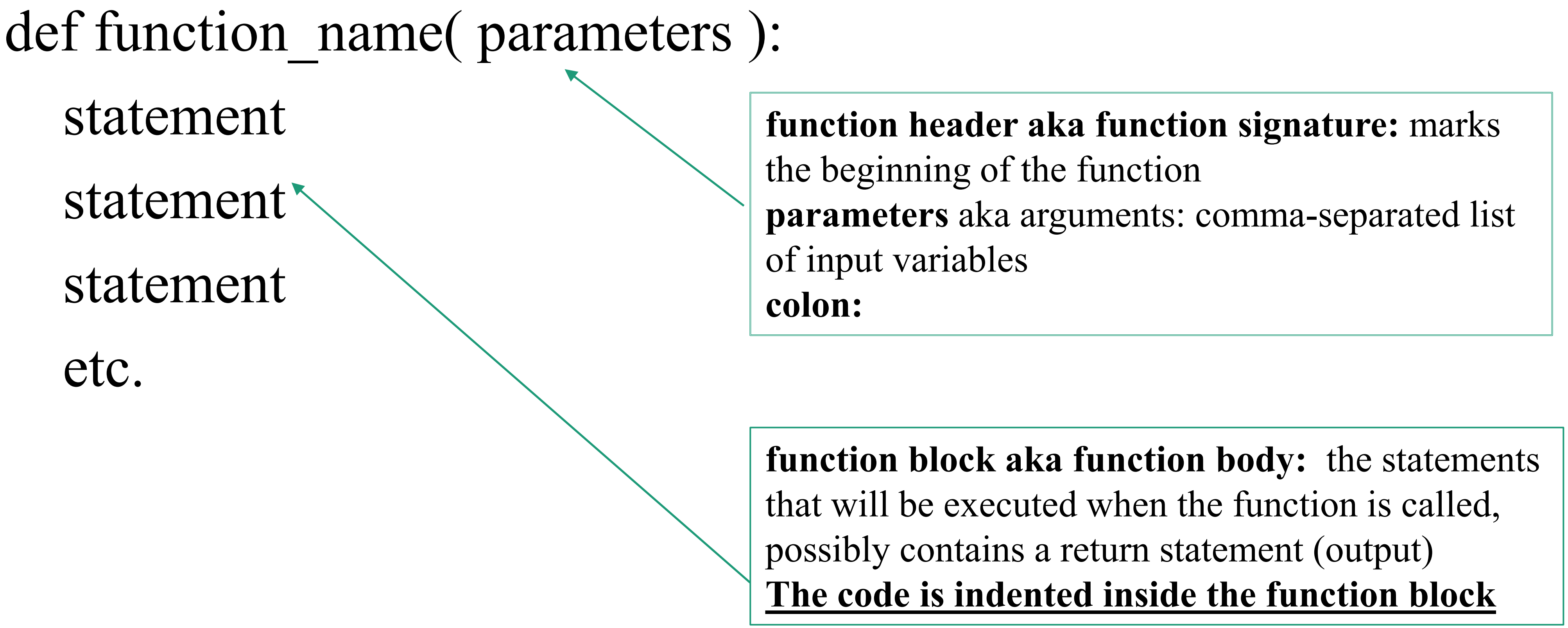
```
def my_Function():
    print('Hello Python')
```

- Call a function
  - We call the function by using the function name, parentheses and arguments in an expression.



```
def my_Function():
    print('Hello Python')

my_Function()
```

Function General Syntax (Summary)



```
def function_name( parameters ):
    statement
    statement
    statement
    etc.
```

function header aka function signature: marks the beginning of the function
parameters aka arguments: comma-separated list of input variables
colon:

function block aka function body: the statements that will be executed when the function is called, possibly contains a return statement (output)
The code is indented inside the function block

Function General Syntax: Example

```
def print_name_uppercase(first_name, last_name):
    print(first_name.upper(), end=" ") # may or may not be ok; ok if first_name is a string only
    print(str(last_name).upper()) # ok to use upper() for sure, since last_name is definitely a string

print_name_uppercase("tiGeR", "woODs")
print_name_uppercase("tiGeR", 5)  # ok; the function changes 5 to a string, which has an upper()
print_name_uppercase(5, "woods")  # crash; 5 is an integer; integers don't have an upper() method
```

Good coding style is shown above:

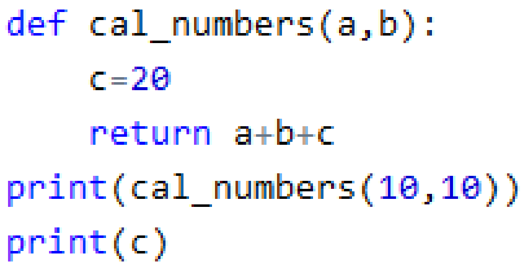
1. Function is defined first, before using it (i.e., defined “above” where it is called)
2. Function is named as a verb
3. Function has clear parameter names
4. Function is indented inside it

Function Coding Style

- Function name should be descriptive and starts with a lowercase character, use snake case if the function name consists of more than one word (i.e., lower_snake_case).
- Example get_data(), display_result(), multiply_numbers()
- Function body must be indented from the function header (signature). This is **IMPORTANT!**
- By default indentation is four spaces. (TAB)

Function Variants

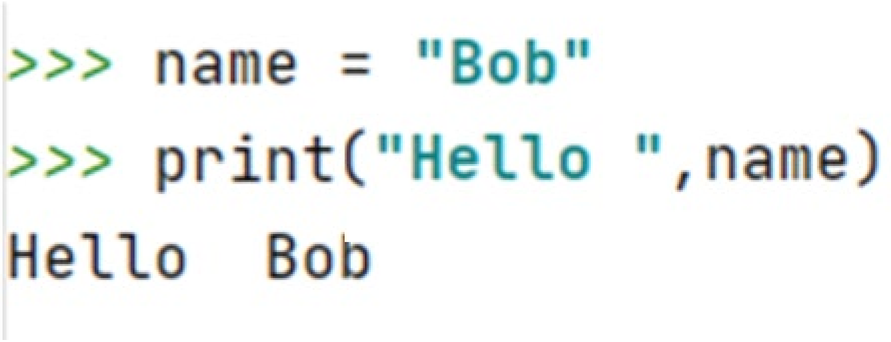
- Functions can be built-in or user defined.
- Built in functions are functions that come with Python standard library.
- Example: print(), input(), str(), int().
- User-defined functions are defined by ourselves to do a certain specific task.
- Functions may return 0 or 1 values.
- Functions that return a value are immediately ended at the “return” statement; you can’t have code after a return statement.
- Functions may or may not receive parameters (arguments).



```
def cal_numbers(a,b):
    c=20
    return a+b+c
print(cal_numbers(10,10))
print(c)
```

Void Functions

- Void function is a function that does not return a value, it simply executes its statements then terminates.
- print() is an example of a void function.
- Function print displays the message and does not return any value.

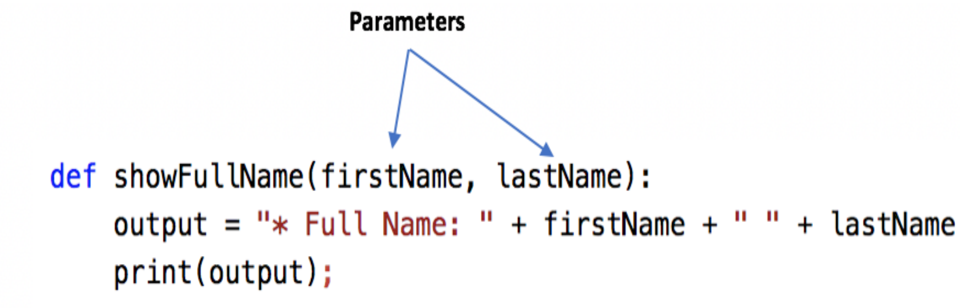


```
>>> name = "Bob"
>>> print("Hello ",name)
Hello  Bob
```

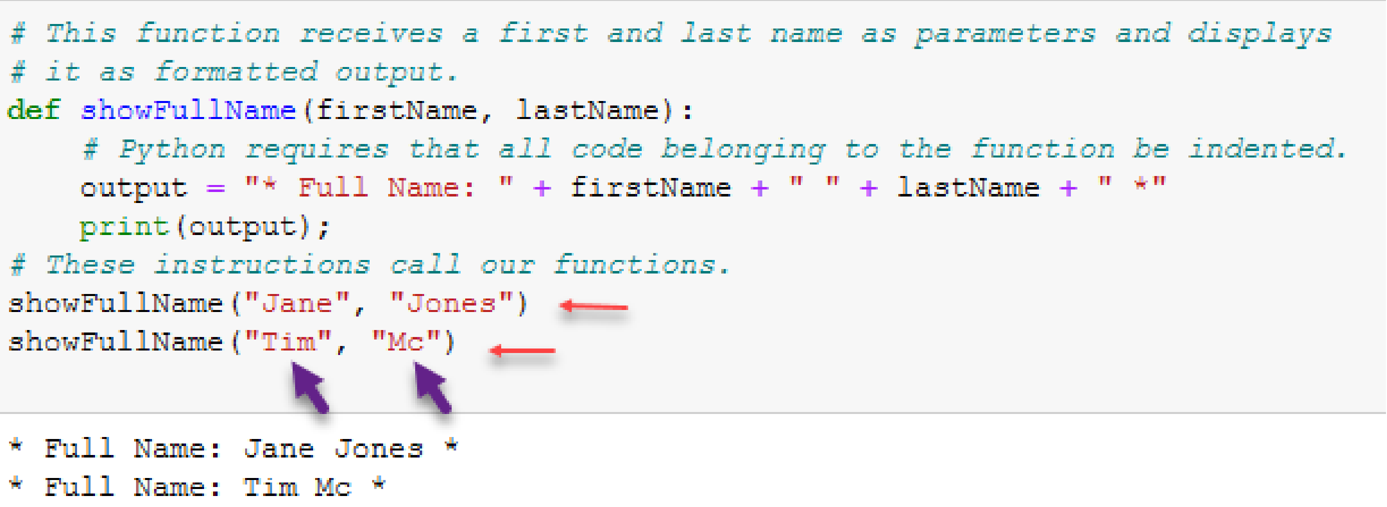
Characteristics of Void Functions:

- No Return Value: They do not use the return statement or return None explicitly.
- Side Effects: They often perform operations that have side effects, like modifying variables or printing to the console.

Arguments/Parameters



```
def showFullName(firstName, lastName):
    output = "* Full Name: " + firstName + " " + lastName
    print(output);
```



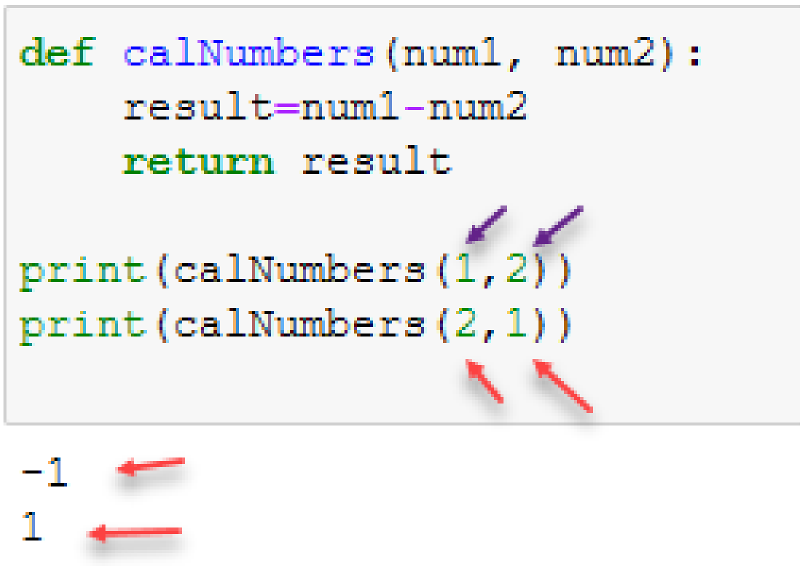
```
# This function receives a first and last name as parameters and displays
# it as formatted output.
def showFullName(firstName, lastName):
    # Python requires that all code belonging to the function be indented.
    output = "* Full Name: " + firstName + " " + lastName + " *"
    print(output);
# These instructions call our functions.
showFullName("Jane", "Jones")
showFullName("Tim", "Mc")
```

```
* Full Name: Jane Jones *
* Full Name: Tim Mc *
```

- An argument is a value which will be passed into the function as its input when we call the function
- The function can do different kinds of work when we modify the arguments

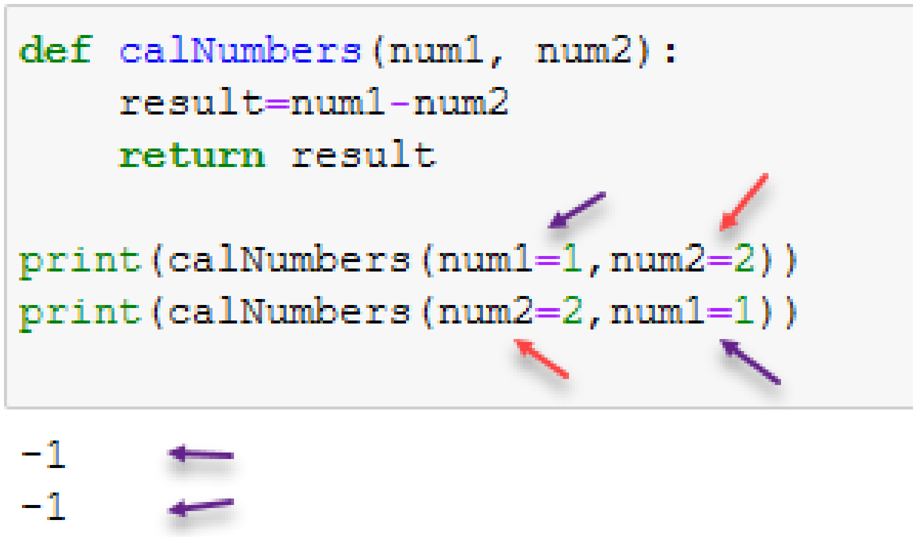
Positional & Keyword Arguments

- *Positional arguments* are arguments that need to be included in the proper position or order.
- A *keyword argument* is an argument passed to a function or method which is preceded by a *keyword* and an equal sign (=).



```
def calNumbers(num1, num2):
    result=num1-num2
    return result

print(calNumbers(1,2))
print(calNumbers(2,1))
```

```
-1
1
```



```
def calNumbers(num1, num2):
    result=num1-num2
    return result

print(calNumbers(num1=1,num2=2))
print(calNumbers(num2=2,num1=1))
```

```
-1
-1
```

Functions

- Generally, you need to get back the result of the function.
- How? You need a return statement.

```
def addTwoNumbers( operandA, operandB ):
   result = operandA + operandB;
   return result # This return statement exits the function and gives a value
                          # to the calling instruction.
sum = addTwoNumbers(3,4) # This is the calling instruction.
print("The result from adding 3 and 4 is " + str(sum))
```

Practice:

Celsius→ Fahrenheit (5)

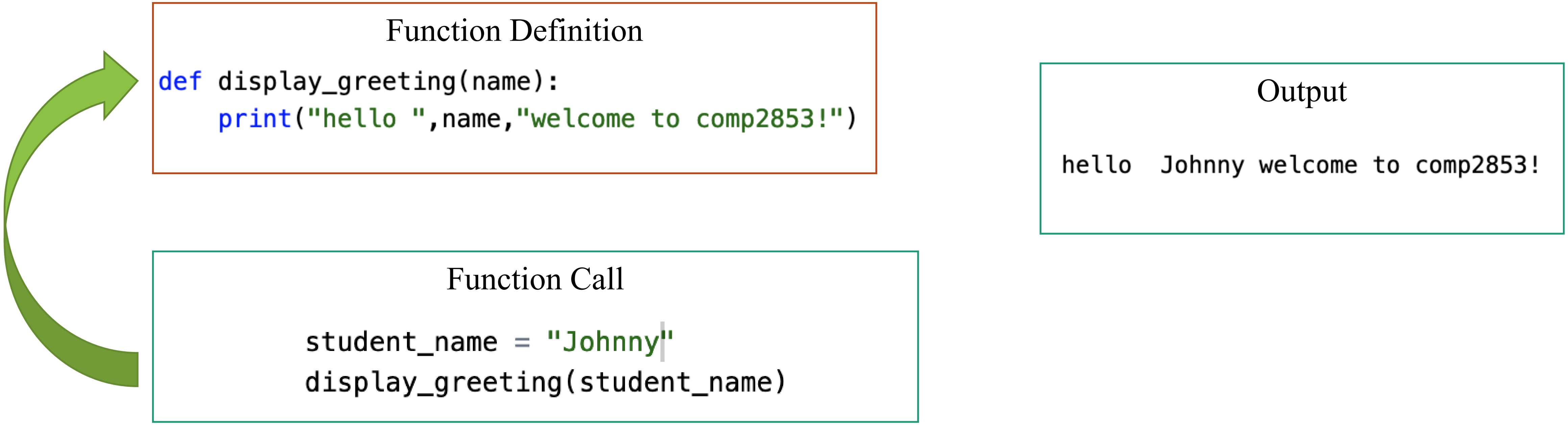
Formula: (Celsius * 1.8)+32=Fahrenheit

Calling a Function

To call a function, it must be defined FIRST; then simply place the function name followed by ().

Pass any parameters if required.

Example:



Function Definition

```
def display_greeting(name):
    print("hello ",name,"welcome to comp2853!")
```

Function Call

```
student_name = "Johnny"
display_greeting(student_name)
```

Output

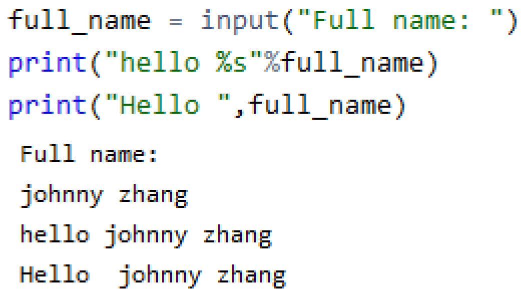
```
hello  Johnny welcome to comp2853!
```

Value Returning Function

Value-returning function executes any statements then returns a value to the point where it was called.

Often, the returned value is assigned to a variable to be further used in the script.

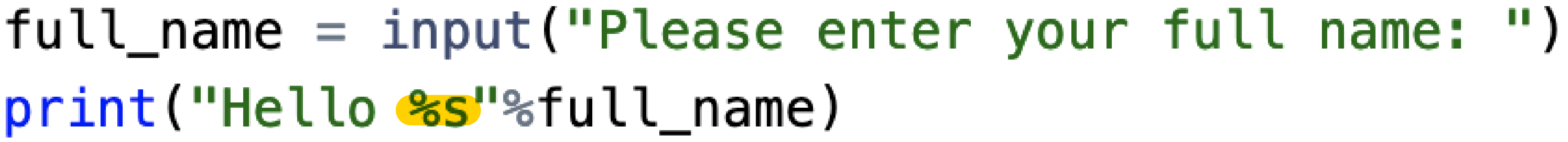
input() is an example of a value-returning function.



```
full_name = input("Full name: ")
print("hello %s"%full_name)
print("Hello ",full_name)
```

```
Full name:
johnny zhang
hello johnny zhang
Hello  johnny zhang
```

Example:



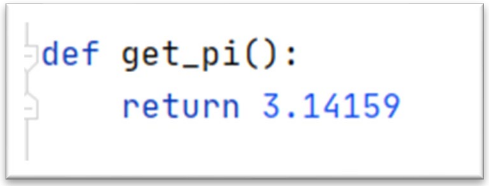
```
full_name = input("Please enter your full name: ")
print("Hello %s"%full_name)
```

Value Returning Function

Return also terminates that function.

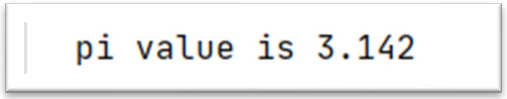
Example:

Function Definition



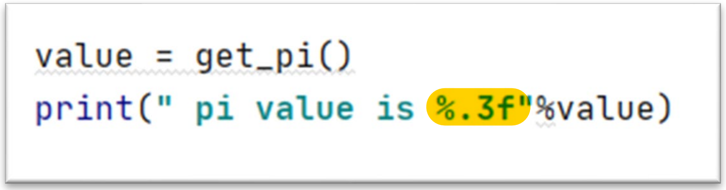
```
def get_pi():
    return 3.14159
```

Output

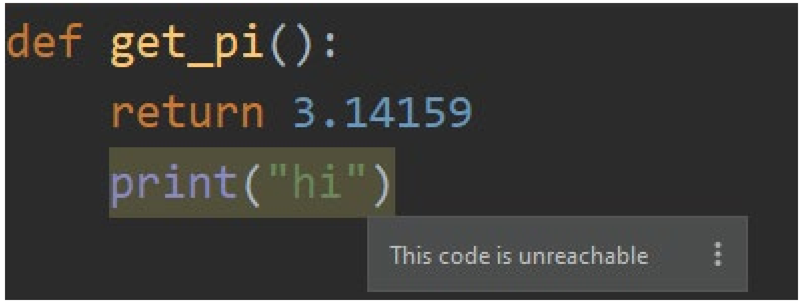


```
pi value is 3.142
```

Function Call



```
value = get_pi()
print(" pi value is %.3f"%value)
```



```
def get_pi():
    return 3.14159
    print("hi")
```

This code is unreachable

Flow of Control

- A function must be defined before it is first called.
- In order to ensure that a function is defined before its first use, you have to know the order in which statements are executed, which is called the *flow of execution*.
- Execution typically begins at the first statement of the program. Statements are executed one at a time, usually in order from top to bottom.

Flow of Control

- Function *definitions* do not alter the flow of execution of the program but remember that statements inside the function are not executed until the function is called.

- A function call is like a detour in the flow of execution. Instead of going to the next statement,
    - the flow jumps to the body of the function,
    - executes all the statements there,
    - and then comes back to pick up where it left off.

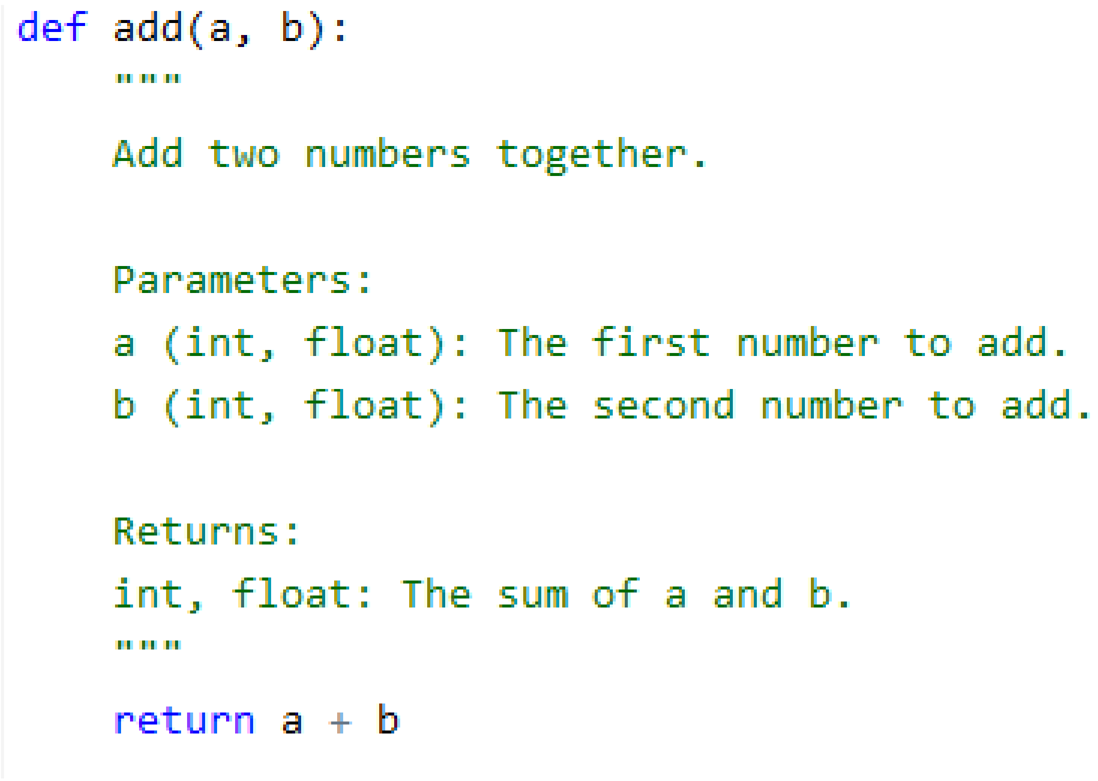
DocString

- A DocString is a more formal comment type, used in functions
- It has several parts:

1. Triple quotation marks
2. A full sentence explaining the function's purpose
3. Explanations in plain language of what each argument is
4. Explanation in plain language of what the function returns

- **Use DocString comments for every function you write.**
- See the next slide, which improves the previous slide's code.

Function Examples with DocString

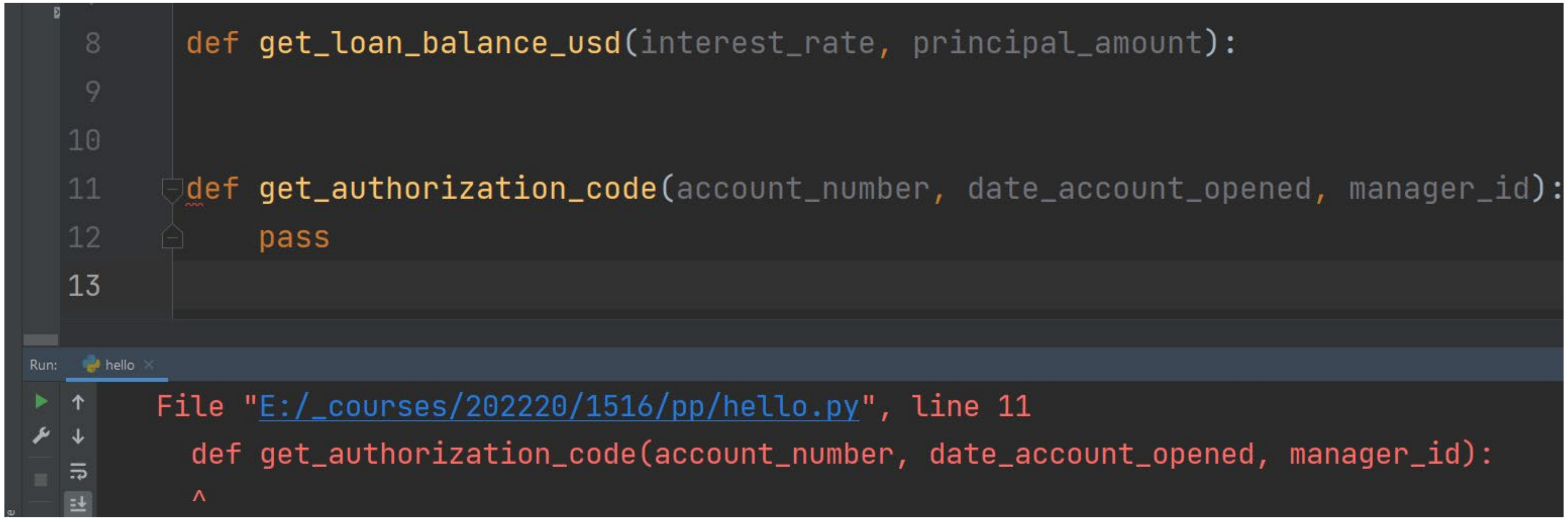


```python
def add(a, b):
    """
    Add two numbers together.

    Parameters:
    a (int, float): The first number to add.
    b (int, float): The second number to add.

    Returns:
    int, float: The sum of a and b.
    """
    return a + b
```

pass

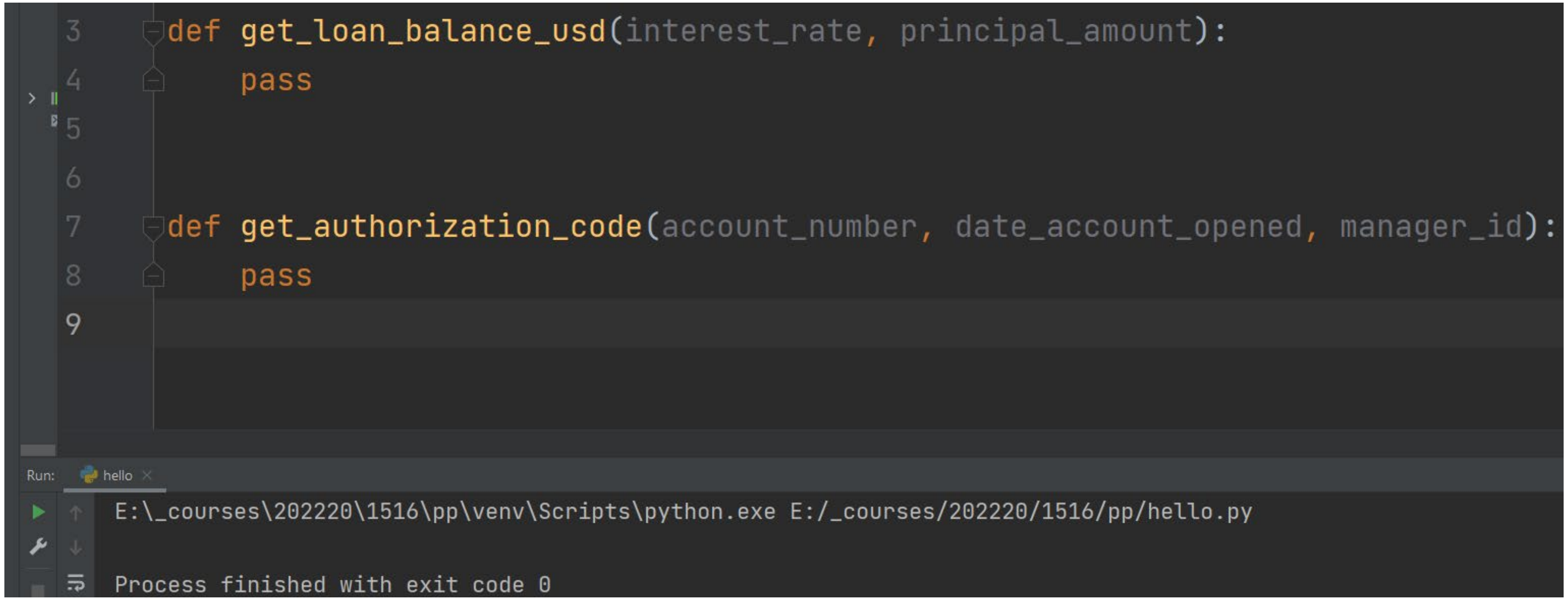
- In Python, the pass statement is a no-operation statement that is often used as a placeholder in situations where syntactically some code is required but no action is needed. It allows you to write empty blocks of code without causing an error.
- Use the keyword **pass** to tell Python “I will fill this in later…just don’t complain that I have an empty function. Leave me alone.”
- See next slide for the difference.

pass



```
 8    def get_loan_balance_usd(interest_rate, principal_amount):
 9
10
11    def get_authorization_code(account_number, date_account_opened, manager_id):
12        pass
13
```

```
File "E:/_courses/202220/1516/pp/hello.py", line 11
  def get_authorization_code(account_number, date_account_opened, manager_id):
  ^
```



```
def get_loan_balance_usd(interest_rate, principal_amount):
    pass


def get_authorization_code(account_number, date_account_opened, manager_id):
    pass

```

```
E:\_courses\202220\1516\pp\venv\Scripts\python.exe E:/_courses/202220/1516/pp/hello.py

Process finished with exit code 0
```

Module vs. Script

- We store our code in Python files.
- A file can be considered either a **module** or a **script**.
- A module is nothing but a collection of functions, classes, and variables. Nothing is running.
- A module can be imported by other files, and the functions can be executed in that other file.
- A script is a Python file that has executing code. It often imports code from modules.
- See next slide.

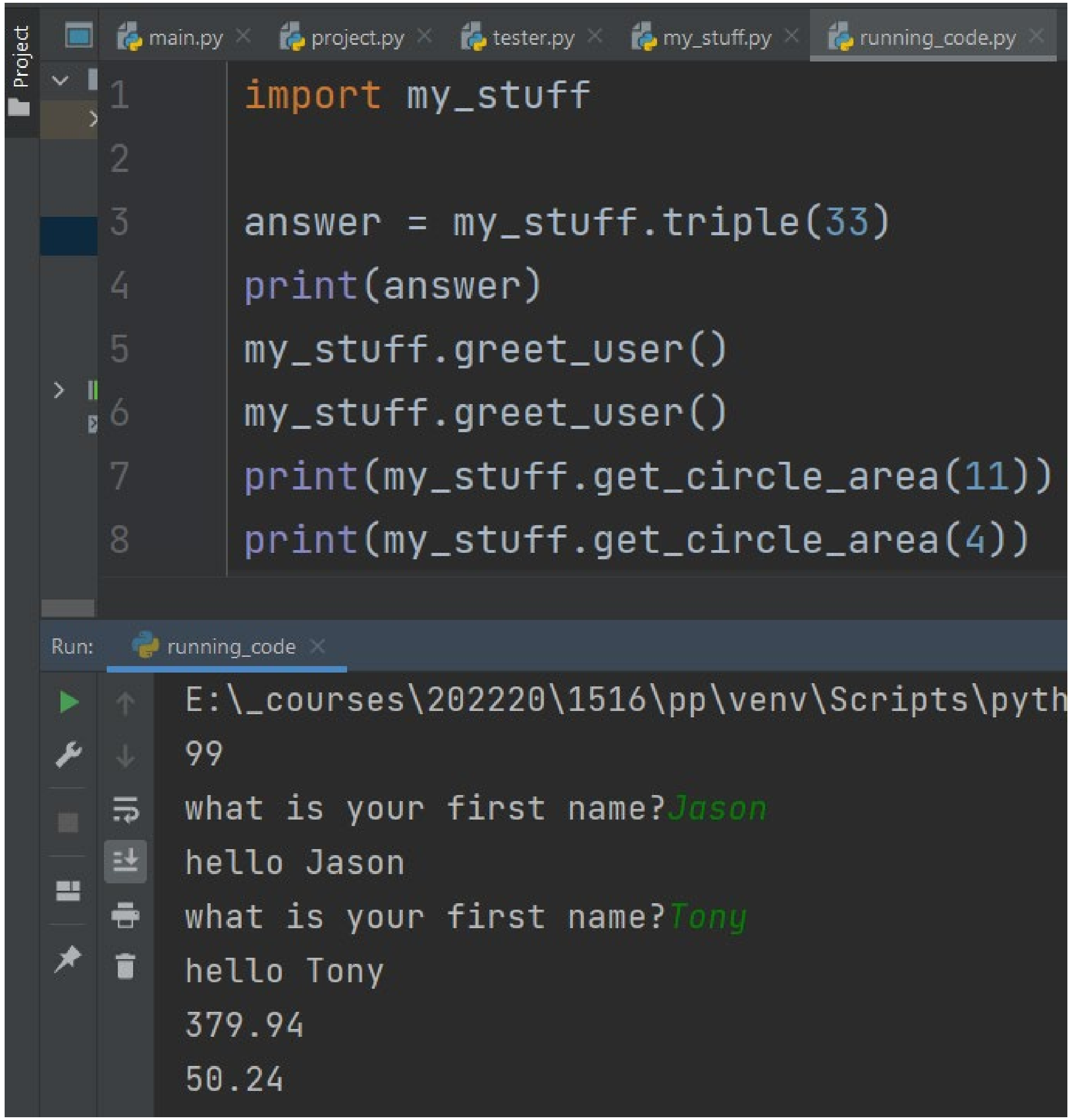
Module: my_stuff.py

```
def triple(num):
    """Takes in a number and returns its triple.
    :param num: the number to be tripled
    :return: the tripled version
    """
    return 3 * num
def greet_user():
    """Asks the user their name, and then tells them hello.
    :return: None
    """
    first_name = input("what is your first name?")
    print("hello", first_name)
def get_circle_area(radius):
    """Calculates and returns the area of the circle with a given radius.
    :param radius: the radius of the circle
    :return: the area of the circle
    """
    return 3.14 * radius ** 2
```

Script: running_code.py

```
from my_stuff import *              # not good
answer = triple(33)

from my_stuff import triple         # ok
answer = triple(33)

import my_stuff                     # great
answer = my_stuff.triple(33)
```

importing



```python
import my_stuff

answer = my_stuff.triple(33)
print(answer)
my_stuff.greet_user()
my_stuff.greet_user()
print(my_stuff.get_circle_area(11))
print(my_stuff.get_circle_area(4))
```

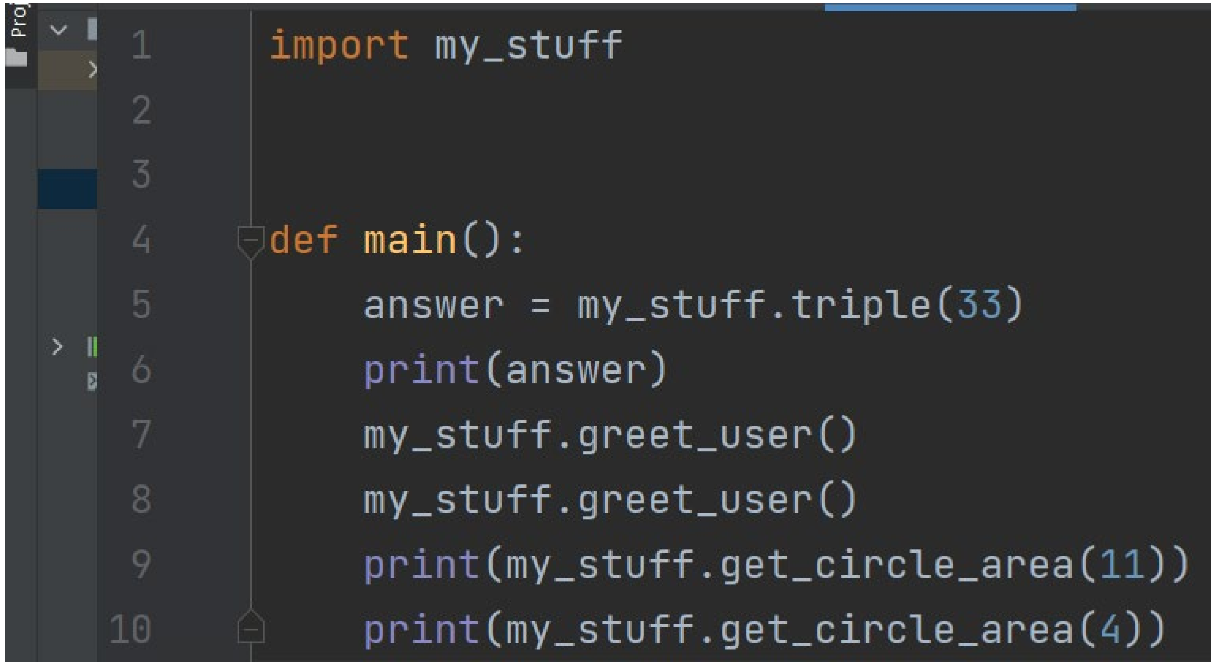
```
E:\_courses\202220\1516\pp\venv\Scripts\pyth
99
what is your first name?Jason
hello Jason
what is your first name?Tony
hello Tony
379.94
50.24
```

Best Practices

- Separate each function from other functions by two blank lines.
- End each file with a blank line.
- Import modules as in the previous two slides.
- Indent function code four spaces.

main

- In many programming languages, including Python, the computer will search for a function named **main()**



```python
import my_stuff


def main():
    answer = my_stuff.triple(33)
    print(answer)
    my_stuff.greet_user()
    my_stuff.greet_user()
    print(my_stuff.get_circle_area(11))
    print(my_stuff.get_circle_area(4))
```

- The **main** function is the entry point for the program, to begin running there
- We should put our code into functions like that, rather than having code simply sitting inside a file

Calling main

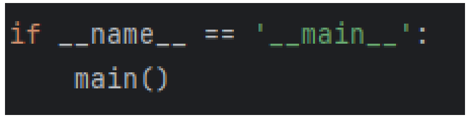
- Now we have created a **main** function, but it is not yet being called anywhere!
- However, we want **main** to be called only for *scripts*.
- We do not want a **module's** main function to be triggered when that module is imported!
- To make sure only a running *script* has its main function called, we put in something called a **main guard**.

main guard

- The main guard is a common Python idiom that ensures code is only executed when a script is run directly, not when it is imported as a module. This is achieved using the if __name__ == "__main__": construct.

- Comparison:

- We can end our file with one of two choices:

  1. main()
  2. 

```
if __name__ == '__main__':
    main()
```

- If we used #1, then main() is called *all the time*, even if this file is simply being imported.

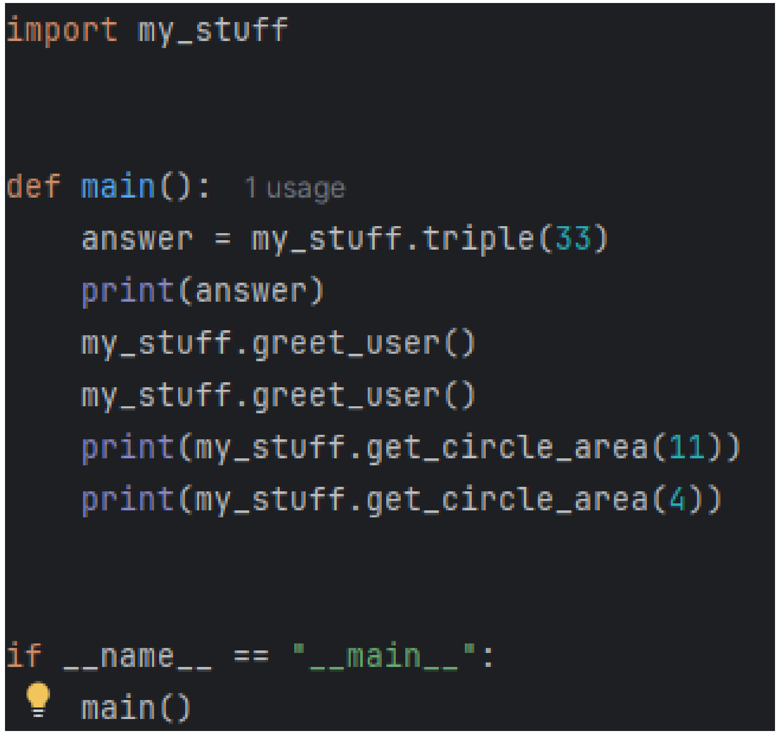
- If we use #2, Python ensures that main() is only called if the script is executed directly, not if it is imported elsewhere.

- Use method #2.

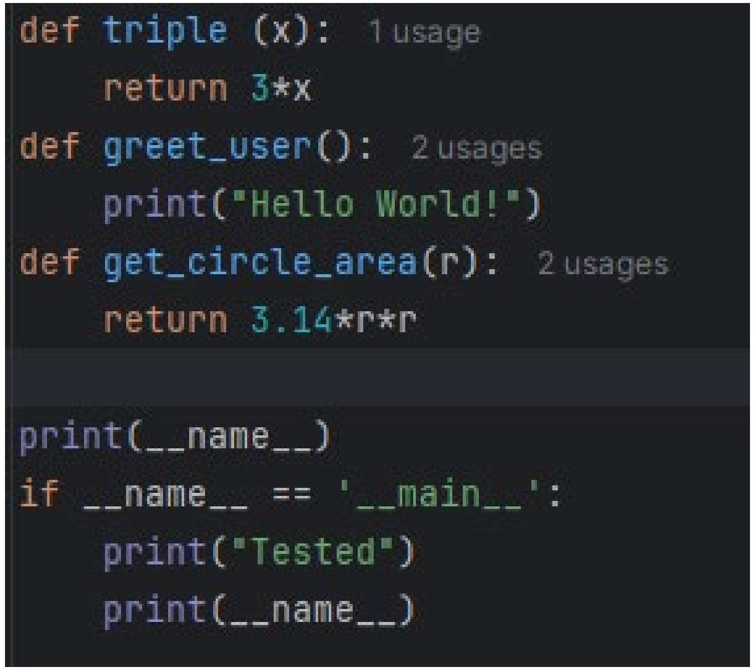
Note:

The if __name__ == '__main__': construct is a common Python idiom that allows you to determine if a Python file is being run as a script or imported as a module.

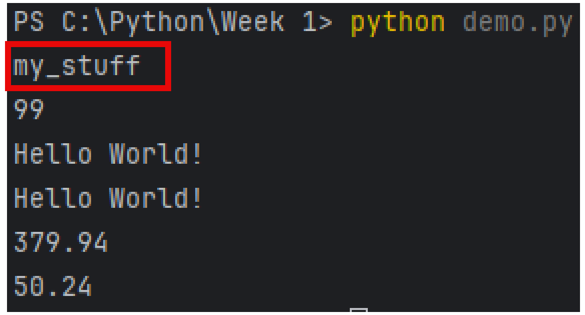
Demo



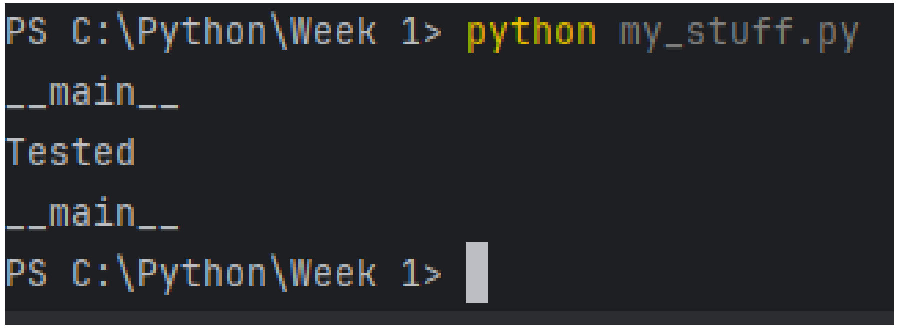
```
import my_stuff


def main():  1 usage
    answer = my_stuff.triple(33)
    print(answer)
    my_stuff.greet_user()
    my_stuff.greet_user()
    print(my_stuff.get_circle_area(11))
    print(my_stuff.get_circle_area(4))


if __name__ == "__main__":
    main()
```



```
def triple (x):  1 usage
    return 3*x
def greet_user():  2 usages
    print("Hello World!")
def get_circle_area(r):  2 usages
    return 3.14*r*r

print(__name__)
if __name__ == '__main__':
    print("Tested")
    print(__name__)
```

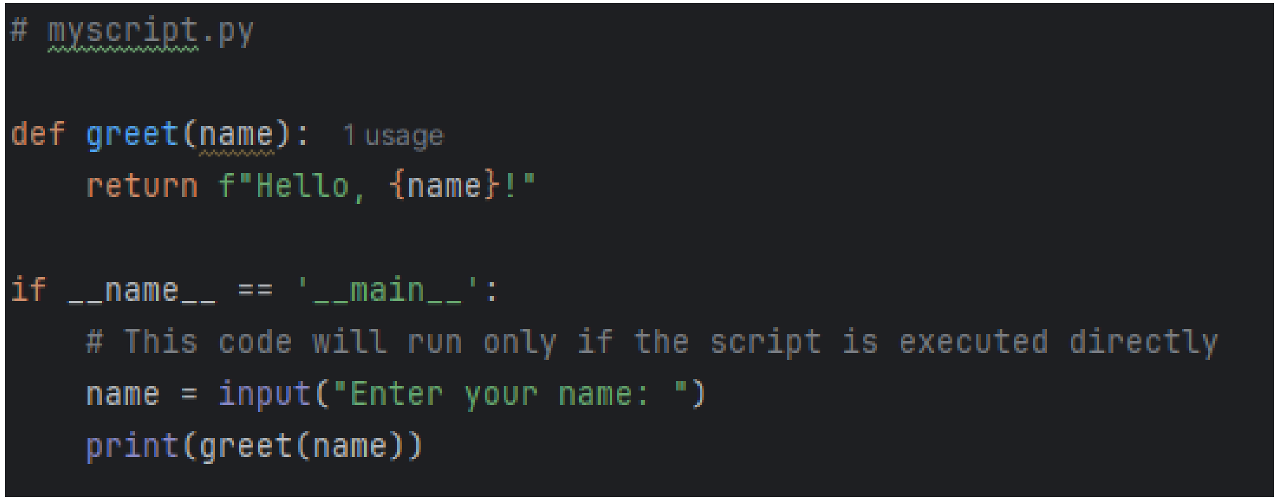


```
PS C:\Python\Week 1> python demo.py
my_stuff
99
Hello World!
Hello World!
379.94
50.24
```

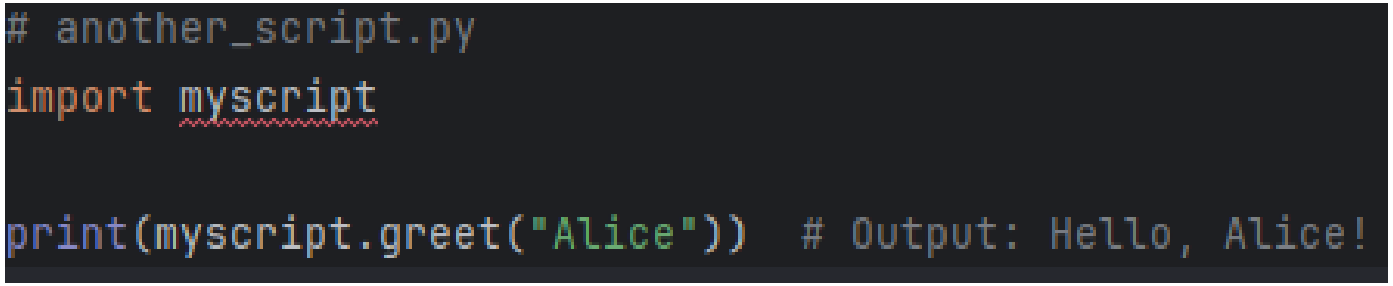


```
PS C:\Python\Week 1> python my_stuff.py
__main__
Tested
__main__
PS C:\Python\Week 1>
```

In Python, __name__ is a special built-in variable that holds the name of the current module. When a Python script is run, __name__ is set to "__main__" if the script is executed directly. If the script is imported as a module into another script, __name__ is set to the name of the module.



```
# myscript.py

def greet(name):  1 usage
    return f"Hello, {name}!"

if __name__ == '__main__':
    # This code will run only if the script is executed directly
    name = input("Enter your name: ")
    print(greet(name))
```



```
# another_script.py
import myscript

print(myscript.greet("Alice"))  # Output: Hello, Alice!
```

Summary:

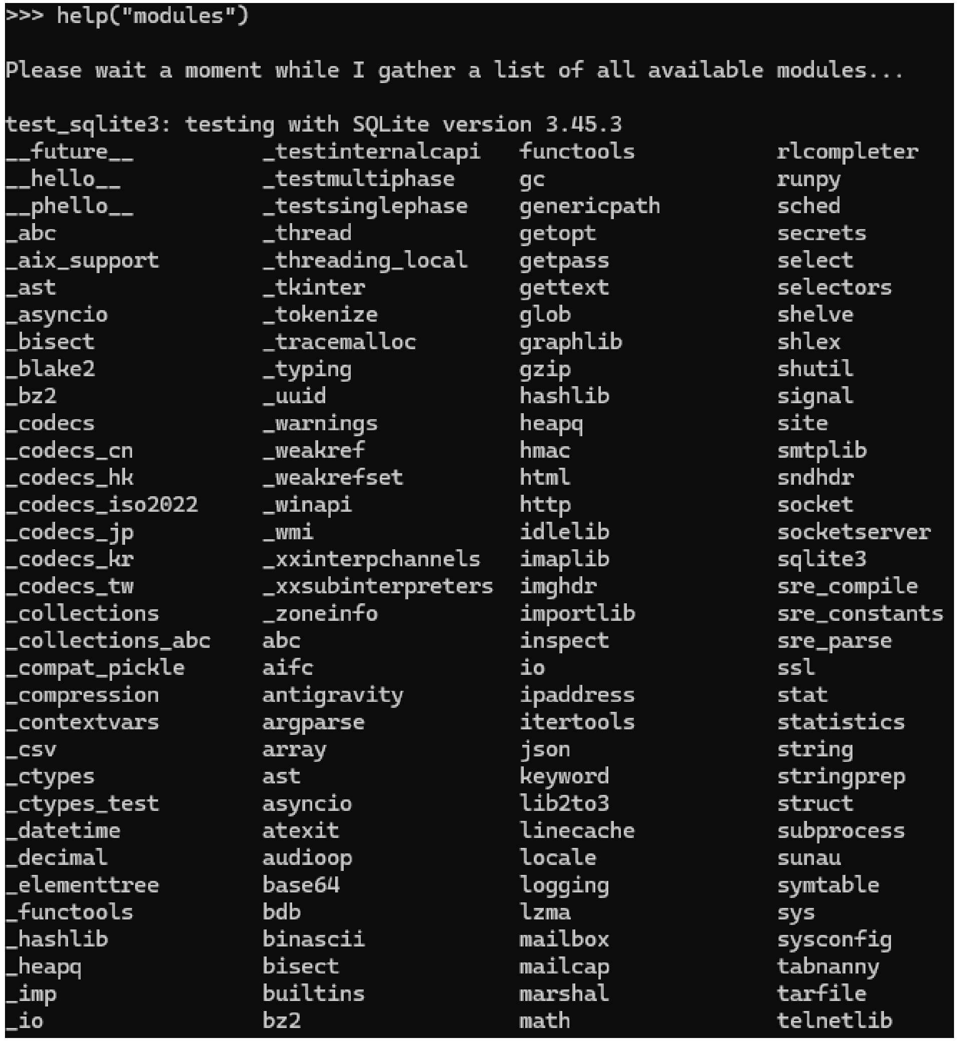
Using if __name__ == '__main__': is a best practice in Python programming that helps you control the execution flow of your scripts and enhances code modularity and reusability.

imports

- We will import other modules soon too
- Python provides a large number of pre-defined functions in a large number of pre-defined modules that are available as a part of Python library.

Built-In Python Modules

To display a list of all available modules, use the following command in Python console.:

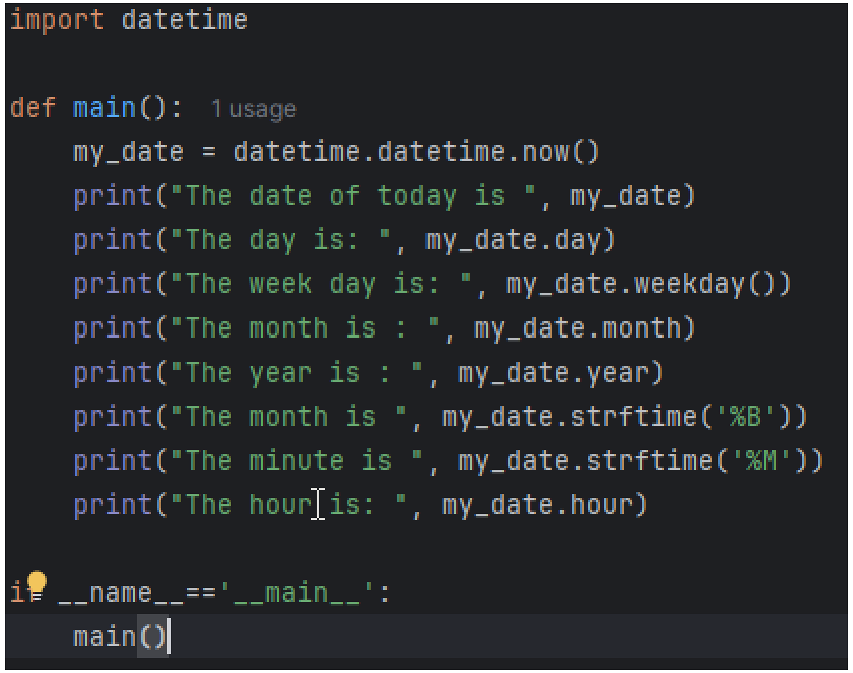


```
>>> help("modules")

Please wait a moment while I gather a list of all available modules...

test_sqlite3: testing with SQLite version 3.45.3
__future__          _testinternalcapi   functools           rlcompleter
__hello__           _testmultiphase     gc                  runpy
__phello__          _testsinglephase    genericpath         sched
_abc                _thread             getopt              secrets
_aix_support        _threading_local    getpass             select
_ast                _tkinter            gettext             selectors
_asyncio            _tokenize           glob                shelve
_bisect             _tracemalloc        graphlib            shlex
_blake2             _typing             gzip                shutil
_bz2                _uuid               hashlib             signal
_codecs             _warnings           heapq               site
_codecs_cn          _weakref            hmac                smtplib
_codecs_hk          _weakrefset         html                sndhdr
_codecs_iso2022     _winapi             http                socket
_codecs_jp          _wmi                idlelib             socketserver
_codecs_kr          _xxinterpchannels   imaplib             sqlite3
_codecs_tw          _xxsubinterpreters  imghdr              sre_compile
_collections        _zoneinfo           importlib           sre_constants
_collections_abc    abc                 inspect             sre_parse
_compat_pickle      aifc                io                  ssl
_compression        antigravity         ipaddress           stat
_contextvars        argparse            itertools           statistics
_csv                array               json                string
_ctypes             ast                 keyword             stringprep
_ctypes_test        asyncio             lib2to3             struct
_datetime           atexit              linecache           subprocess
_decimal            audioop             locale              sunau
_elementtree        base64              logging             symtable
_functools          bdb                 lzma                sys
_hashlib            binascii            mailbox             sysconfig
_heapq              bisect              mailcap             tabnanny
_imp                builtins            marshal             tarfile
_io                 bz2                 math                telnetlib
```

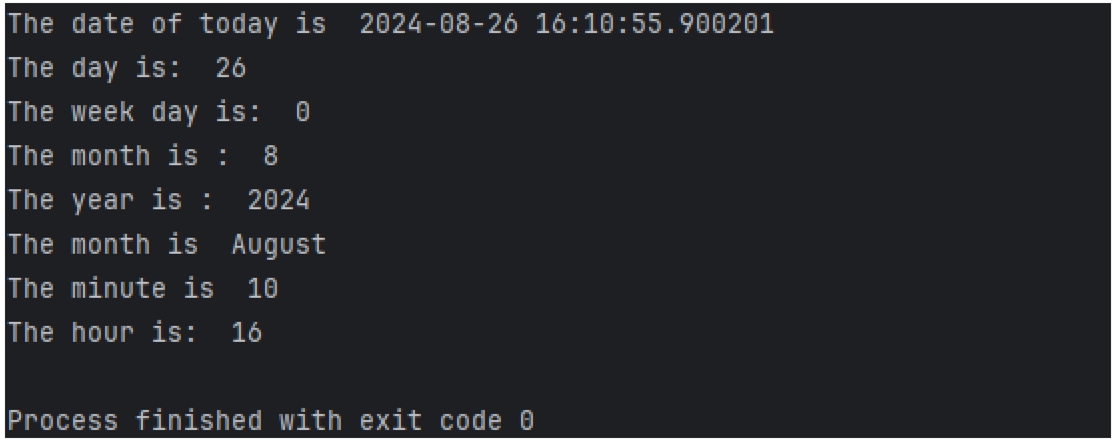
Documentation for the built-in Python modules is available at: https://docs.python.org/3/library/

Built-In Python Modules: **datetime Module**

- The datetime module in Python provides classes for manipulating dates and times. It's part of the standard library, so you don't need to install anything extra to use it.



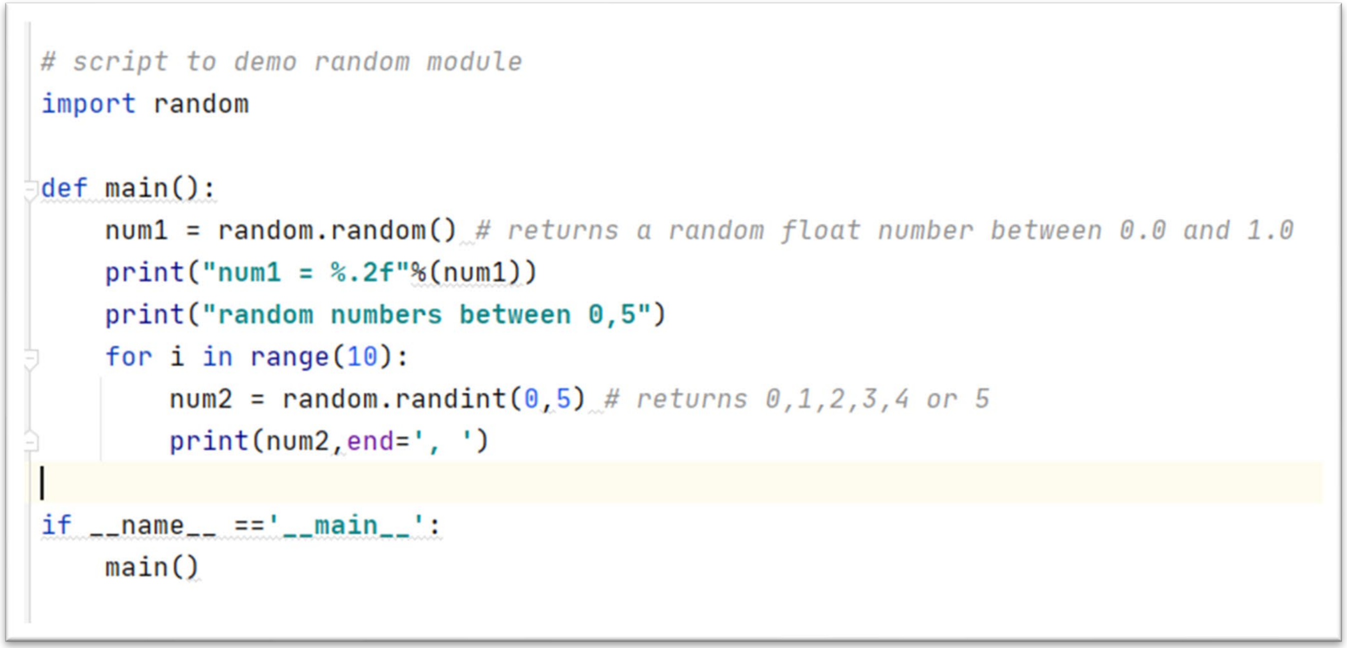
```
import datetime

def main():
    my_date = datetime.datetime.now()
    print("The date of today is ", my_date)
    print("The day is: ", my_date.day)
    print("The week day is: ", my_date.weekday())
    print("The month is : ", my_date.month)
    print("The year is : ", my_date.year)
    print("The month is ", my_date.strftime('%B'))
    print("The minute is ", my_date.strftime('%M'))
    print("The hour is: ", my_date.hour)

if __name__=='__main__':
    main()
```



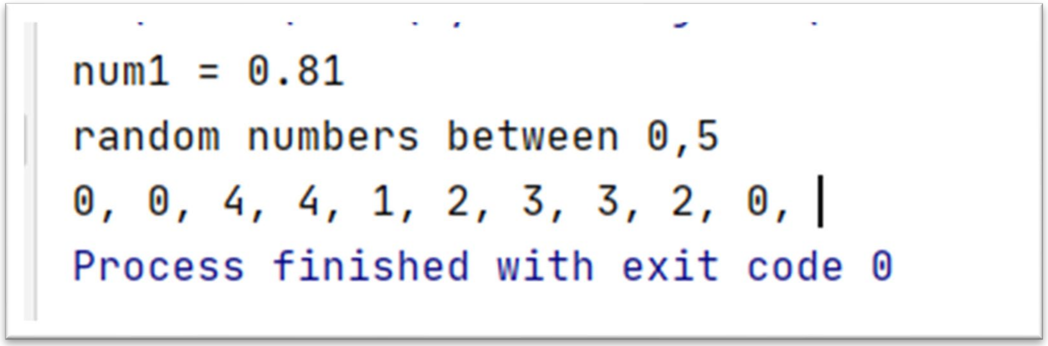
```
The date of today is  2024-08-26 16:10:55.900201
The day is:  26
The week day is:  0
The month is :  8
The year is :  2024
The month is  August
The minute is  10
The hour is:  16

Process finished with exit code 0
```

Built-In Python Modules: **random Module**

- Random module allows the user to generate random numbers.



```
# script to demo random module
import random

def main():
    num1 = random.random() # returns a random float number between 0.0 and 1.0
    print("num1 = %.2f"%(num1))
    print("random numbers between 0,5")
    for i in range(10):
        num2 = random.randint(0,5) # returns 0,1,2,3,4 or 5
        print(num2,end=', ')

if __name__ =='__main__':
    main()
```



```
num1 = 0.81
random numbers between 0,5
0, 0, 4, 4, 1, 2, 3, 3, 2, 0, 
Process finished with exit code 0
```