Lecture 7: User-Defined Functions

- Why Use Functions
- Parameters, arguments, returning results
- Local and Global Scope
- Where to Place Function Definitions
- Functions - Python docstring
- Function default arguments
- Using main()

Why Use Functions

- Functions provide (black boxes of) functionality
  - Reusable code
  - Easier to debug
  - Easier to read/maintain
- Named sequence of statements that performs some useful operation. May **take arguments** and may **produce a result**.
- Most programming languages support a form of user-defined functions. However, they may be referred to as Subroutines, Procedures, Methods, Subprograms.

Python Functions

- Python built-in functions. E.g.,

  `print(), input(), str(), int().`

- We will now create user-defined functions.
- In this section:
  - How to define a function
  - How to call a function
  - Function arguments/parameters
  - Return an explicit value from a function.

Syntax

```
def function_name(parameter1, parameter2, ... parameterN):
    function_block
```

- A Python function is defined using the **`def`** keyword, followed by the **function name**, followed by parentheses **()** ending with a **colon**.
- Parameters are optional. The parentheses **()** may contain a list of parameters (as shown above) or be empty (as below).

```
def function_name():
    function_block
```

- The code to be executed by the function is indented.

Function with zero parameters

```
def greet1():
    print("Hello")
```

- The function will only perform an operation when the function is 'called'.
- This function can be called by writing the function name followed by empty parentheses:

```
# main program
greet1()                    # Hello
```

Functions with parameters

- Functions can be defined to expect zero, one or more arguments .

```
def greet2(name):
   print("Hello " + name)


def mysum(a, b):
   print(a + b)
```

- When the function is called, we also pass data (arguments) to the function

```
greet2("Westminster")        # Hello Westminster
greet2("Python")             # Hello Python
mysum(3, 4)                  # 7
```

Function with return value

```
# function definition
def mysum(a, b):
    return a + b


#function call - returned value stored in variable
result = mysum(3, 4)
print(result)
```

- A function that returns a value would normally assign the value to a variable or use it as part of an expression otherwise the return value is lost.

printing vs return

```
def mysum(a, b):
   print(a + b)

mysum(3, 4)    # call function
```

- Value of 7 is printed from within the function but is not available to work with later.

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

result = mysum(3, 4)   # call function
```

- We have a value stored in variable result and it is available to work on later in the program.

Multiple return values

- When you need to return more than one piece of data, you would return a collection of data:

```
def returnMultiple():
    a = 5
    b = 10
    return [a, b]     # data type holding multiple items
```

- What type of collection would the following return?

```
return a, b
```

Argument vs Parameter

```
def mysum(a, b):
    print(a + b)

mysum(3, 4)    # call function
```

- **argument:** A value passed to a function when a function is called.
- **parameter***:* the variable listed inside the parentheses in the function definition - used to refer to the value passed to it.
- Function assigns the arguments **3** and **4** to the parameters **a** and **b**.

Functions without return

- A function without a return statement is know as void, and they return **None,** Python's special object for "nothing".

```
def print42():
    print(42)

a = print42()          # 42
print(a)               # None
```

Global vs Local variables

- If a variable is defined outside of any function, it is a **global** variable.
- If a variable is defined anywhere within a function, it is a **local** variable.

```
def square(x):
    y = x * x
    return y

z = square(10)
print(y)   # Error
```

- Variable **y** only exists while the function is being executed (its lifetime).
- Parameters are also local - lifetime of **x** begins when square() is called, and ends when the function completes its execution.

Global vs Local variables

- What is the output?

```
def b():
    a = 99          # local variable a
    print(a)

a = 42              # global variable a
print(a)
b()
print(a)
```

Recap

- When passing variables into a function, you're passing the value of that variable (not the variable itself.) This will not alter the variable *n* outside of the function.

```
n = 5
def changeNum(n):
    n += 5
    print(n += 5)    #10

changeNum(n)
print(n)    # 5
```

- However, see the example on the next slide…..

In-Place Algorithms

- Changing information via index is different. An index works via memory location and not by reference. Changing a list item in a list by the index location will alter the original variable.

```
sports = [ "baseball", "football", "hockey", "basketball" ]
def change(aList):
      aList[ 0 ] = "soccer“

print(sports)   # ['baseball', 'football', 'hockey', 'basketball']
change(sports)
print(sports)    # ['soccer', 'football', 'hockey', 'basketball']
```

Where to Place Function Definitions

```
res = return_sum(4,5)

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

- This will cause: **NameError: name 'return_sum' is not defined**
- You need to define your function before you call it.
- Put your functions at the start of programs, below any import statements, to avoid issues.

pass statement

- The pass statement in Python is used as a temporary placeholder for future code. E.g., for a Python function that will be fully implemented at a later time.
- Nothing happens when it executes, but it avoids errors where empty code is not allowed.
- Empty code is not allowed in **function definitions**, loops and if statements.

```
def my_function():
    pass
```

Self-Check Question 1 & 2

1. Which of the following is a valid function header (first line of a function definition)?

    A. def drawCircle(t):
    B. def drawCircle:
    C. drawCircle(t, sz):
    D. def drawCircle(t, sz)

2. Write a function that **returns the maximum of two numbers.**

Functions - Python docstring

- At the start of a function, you may write a (multiline) **docstring** to explain what the function does.
- Using triple quotes (either type) allows newlines within the docstring.

```
def mysum(a,b):
    """ Return the sum of parameters a and b.
    Last modified 24/09/2020 """
    return a + b
```

- Now the docstring should appear in the 'help' for the function.

```
>>> help(mysum)
```

Function default arguments (1)

- Defaults values can be provided. E.g., **inc** is assigned 1 in the parameter list:

```
def my_function(start, stop, inc=1):
    print(f'start= {start}, stop= {stop}, inc= {inc}’)
```

- When calling the function, the third argument is now **optional**. However, the default value can be **overridden**. Place these at the end of the parameter list.

```
a = my_function(1, 10, 5)   # start= 1, stop= 10, inc= 5
a = my_function(1, 10)      # start= 1, stop= 10, inc= 1
```

Function default arguments (2)

- Example 2: If function called without an argument use default value .

```
def my_country(country = "UK"):
    print("I am from " + country)

my_country("Spain")
my_country("India")
my_country()
```

- Note: default parameters **MUST** always go after non-default parameters

Function default arguments (2)

- Example 2: If function called without an argument use default value.

```
def my_country(country = "UK"):
    print("I am from " + country)

my_country("Spain")
my_country("India")
my_country()
```

- Note: default parameters **MUST** always go after non-default parameters

Function default arguments (3)

- You have met default arguments in use before.
- E.g., The print function uses end='\n'as a default value.
  - We can customize the value of end to suppress printing a new line, by using end=''

```
print('Dog', end='‘)
print('Cat')
```

Using main() – example 1

```
def squareit(n):
    return n * n

def cubeit(n):
    return n*n*n

num = int(input("Please enter a number "))
print(squareit(num))
print(cubeit(num))
```

Using main() – example 1

- Many programming languages (e.g. Java and C++) use a function main() that is automatically invoked when a program is executed. Although not required by Python, some programmers incorporate main() in their programs.
- The following three lines are logically related in that they provide the main tasks that the program will perform.
- So we create a function main() with these.

```
def main():
    num = int(input("Please enter a number"))   #1
    print(squareit(num))                        #2
    print(cubeit(num))                          #3
```

```
def squareit(n):
    return n * n

def cubeit(n):
    return n*n*n

def main():
    num = int(input("Please enter a number"))
    print(squareit(num))
    print(cubeit(num))

main()
```

- Remember, in Python there is nothing special about the function name main(). We use main() to be consistent with other languages.

special variable called __name__

- Before the Python interpreter executes a program, it defines a special variable called __name__ and sets the value to:
  - **"__main__" when the program is being executed by itself.**
  - Or the module name if the program is being imported by another program.
- **Therefore, we can use an if statement to check if the program is standalone (value is "__main__")** or being used by another program. Then select to execute code based on value.

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

special variable called __name__

```
def squareit(n):
    return n * n

def main():
    num = int(input("Please enter a number"))
    print(squareit(num))

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

Lecture 7 covered User-Defined Functions

- Why Use Functions
- Parameters, arguments, returning results
- Local and Global Scope
- Where to Place Function Definitions
- Functions - Python docstring
- Function default arguments
- Using main()