Hyperiondev

Beginner Programming with Functions

Objectives

1. Understanding functions
    a. What is a function?
    b. Why functions?
    c. Built-in functions in Python
2. Creating self-defined functions
    a. Keywords to declare a function
    b. Calling functions
    c. Default values
3. Understanding function scope

Lecture – Housekeeping

- The use of disrespectful language is prohibited in the questions, this is a supportive, learning environment for all - please engage accordingly.
- No question is daft or silly - **ask them!**
- There are Q/A sessions midway and at the end of the session, should you wish to ask any follow-up questions.
- You can also submit questions here: http://hyperiondev.com/sbc4-se-questions
- For all non-academic questions, please submit a query: www.hyperiondev.com/support
- Report a safeguarding incident: http://hyperiondev.com/safeguardreporting
- We would love your feedback on lectures: https://hyperionde.wufoo.com/forms/zsgv4m40ui4i0g/

Github Repository - Lecture Examples

https://github.com/HyperionDevBootcamps/C4_SE_lecture_examples

What is a Function?

- Reusable and Organised block of code.
- Sometimes called a ‘method’, although technically methods are associated with the objects of the class they belong to, whereas functions are not associated with any object.
- Similar to functions in maths - f(x) takes input x and produces some output.
- Useful for abstraction
    - For example, “make a cup of tea” vs “boil water, add tea bag, add sugar, add milk, stir”.

Example

```
# Abstraction is used to hide background details or
# any unnecessary implementation about the data
# so that users only see the required information
def makeCupOfTea():
    print("Boil water")
    print("Add tea bag")
    print("Add sugar")
    print("Add milk")
    print("Stir")
```

Why Functions?

- **Reusable code** - Sometimes you need to do the same task over and over again.
- **Error checking/validation** - Makes this easier, as you can define all rules in one place.
- **Divide code up into manageable chunks** - Makes code easier to understand.
- **More rapid application development** - The same functionality doesn't need to be defined again.
- **Easier maintenance** - Code only needs to be changed in one place.

Functions in Python

- Python comes bundled with built-in functions.
- Examples:
  - **print(string)** - prints string to console.
    Eg. print(“Hello World”)
  - **input(string)** - prints string to console, then reads input as string. Eg. num = input(“Please enter a number”)
  - **len(list)** - finds the length of an array.
    Eg. print(len([1,2,4])) # Prints 3
  - **int(data)** - converts the value to an integer.
    Eg. num = int(“5”)

Is that all of the Functions in Python?

- The list of functions that you can use in Python doesn't just stop with what is built in.
- Using Pip (python package manager), you can install various packages containing **modules**.
  - Note: Some packages are already installed by default in Python, such as the maths package.
- These modules can be imported into your script using an import statement.

Importing Modules

- Let's take a look at the maths module. Let's say that you want to use floor(), which rounds a number off to the smallest integer. There are two ways to access this:
- ```
  import math
  my_result = math.floor(my_num)
  ```

- ```
  from math import floor
  my_result = floor(my_num)
  ```

Creating our own Functions

- Uses the def keyword (for define):
  - def add_one(x): # create new function called add_one
    y = x + 1
    return y
- Important keywords:
  - def - tells Python you are defining a function
  - return - if your function returns a value, then use this keyword to return it.

Some Important Terms

- **Function** - A block of code that performs an action.
- **Method** - A function defined on or owned by an object. Not quite the same thing as a function but very similar for our purposes at this stage of learning.
- **Parameters** - The defined input of a function.
- **Arguments** - The values passed to parameters.

Calling Functions

- Functions with one **required positional** input:
  - my_function1(input1)
- Functions with two **required positional** inputs:
  - my_function2(input1, input2)
- Functions with one **required positional** input and one **optional keyword** input:
  - my_function3(input1, keyword_arg=input2)

```python
#Calling functions
def my_function1(input1):
    return input1
def my_function2(input1,input2):
    return input1 + input2
def my_function3(input1, keyword_arg = 2):
    return input1 + keyword_arg

a = my_function1(1)
b = my_function2(1,2)
c = my_function3(1)

print(a)
print(b)
print(c)
```

Default Values

- Remember **optional keyword** arguments? These are made with default values.
- def multiply(num1, num2 = 5):
- This can be called with multiply(10), for example.
- The default value can be overwritten with multiply(10, num2=6).

```
def multiply(num1,num2 = 5):
    sum = num1 * num2
    return sum

answer1 = multiply(10)
answer2 = multiply(10,num2 = 6)

print(answer1)  #prints 50
print(answer2)  #prints 60
```

Scope

- Where is a variable accessible in Python?
- Generally, whenever code is executed, variables become accessible across the entire script.
- Functions are different, however. Variables declared within functions are not accessible outside the function.
  - This avoids variable names being overwritten.

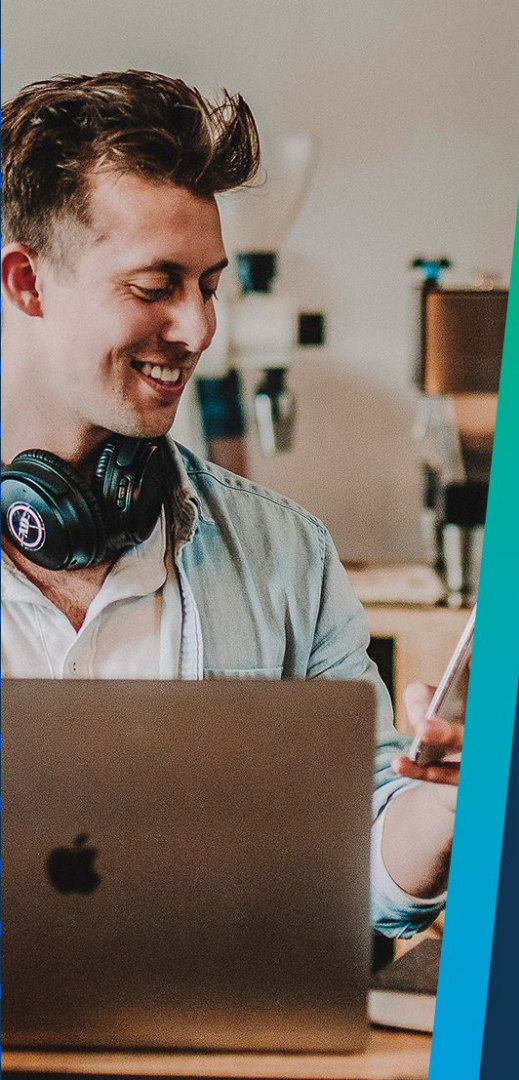
```
def multiply(x,y):
    product = x * y
    return product

answer1 = multiply(2,3)

print(f"{x} times {y} is {answer1}")
```

```
    print(f"{x} times {y} is {answer1}")
NameError: name 'x' is not defined
```

Hyperiondev

Q & A Section

Please use this time to ask any questions relating to the topic explained, should you have any



Hyperiondev

Thank you for joining us