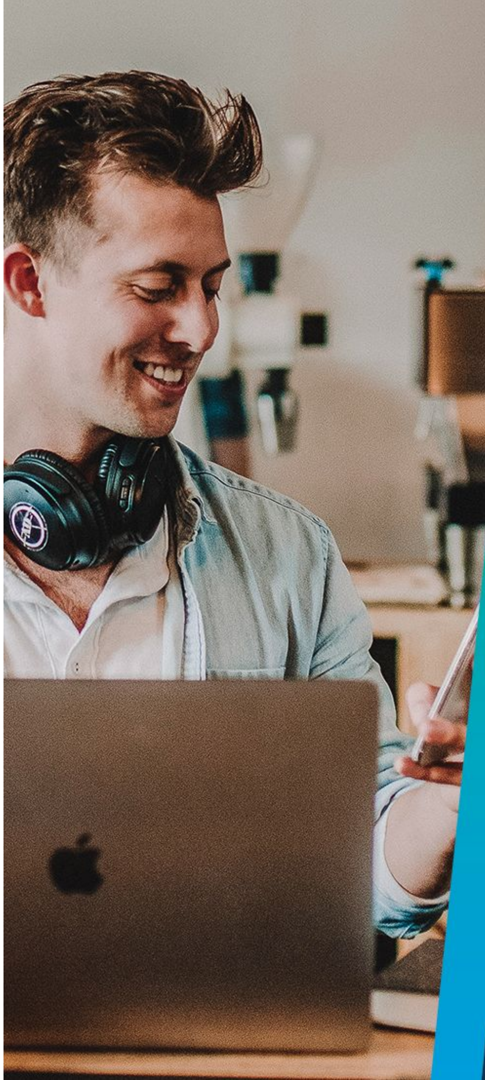


Hyperiondev

Workshop on Functions

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/zsgv4m40ui4i0q/

Hyperiondev

Objectives

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

Github Repository - Lecture Examples

https://github.com/HyperionDevBootcamps/C4_SE_lecture_examples

PEP Documentation

https://docs.python.org/3/library/stdtypes.html

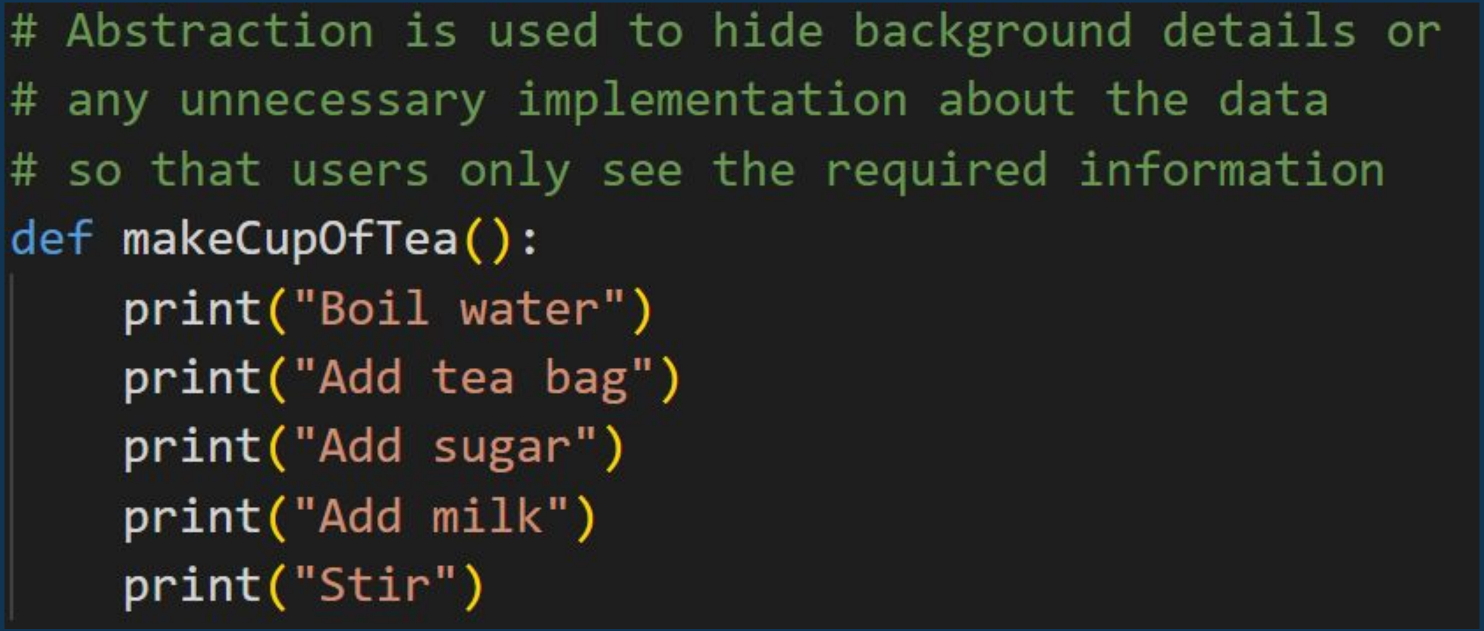
https://docs.python.org/3/library/functions.html

https://docs.python.org/3/library/stdtypes.html#string-methods

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. Eg. $f(x) = x + 1$
- Useful for abstraction
  - For example, “make a cup of tea” vs “boil water, add tea bag, add sugar, add milk, stir”.

Example



```python
# 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")
```

Hyperiondev

Why Functions?

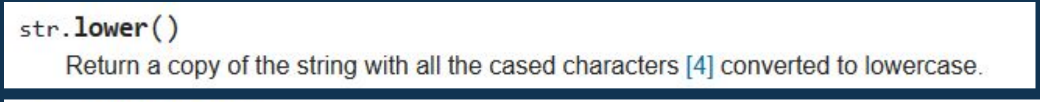
- **Reusable code** - Sometimes you need to do the same task over and over again.
- **Error checking/validation** - Makes this easier, as you can define all rules in one place.
- **Divide code up into manageable chunks** - Makes code easier to understand.
- **More rapid application development** - The 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 a list.
    Eg. print(len([1,2,4])) # Prints 3
  - **int(data)** - converts the value to an integer.
    Eg. num = int("5")

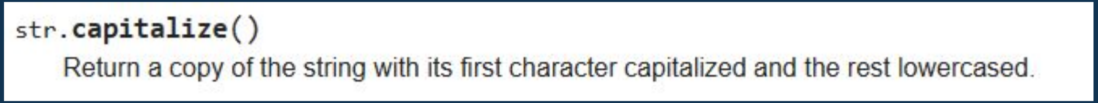
PEP doc: https://docs.python.org/3/library/functions.html

String Functions



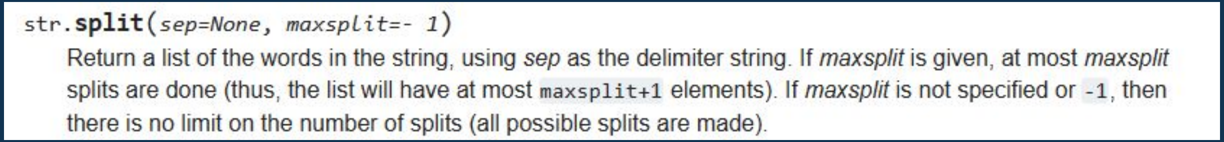
`str.lower()`

Return a copy of the string with all the cased characters [4] converted to lowercase.



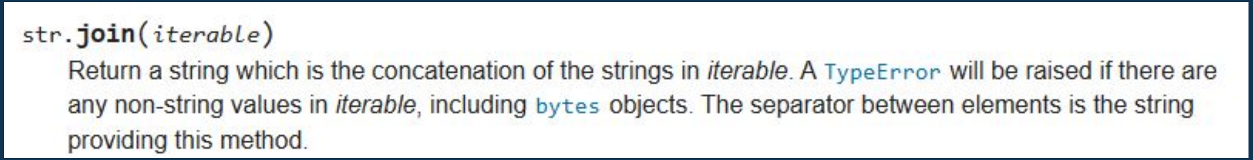
`str.capitalize()`

Return a copy of the string with its first character capitalized and the rest lowercased.



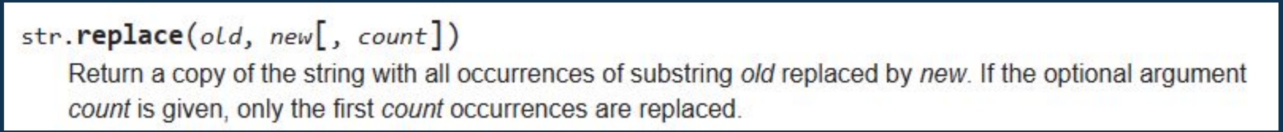
`str.split(sep=None, maxsplit=- 1)`

Return a list of the words in the string, using *sep* as the delimiter string. If *maxsplit* is given, at most *maxsplit* splits are done (thus, the list will have at most `maxsplit+1` elements). If *maxsplit* is not specified or `-1`, then there is no limit on the number of splits (all possible splits are made).



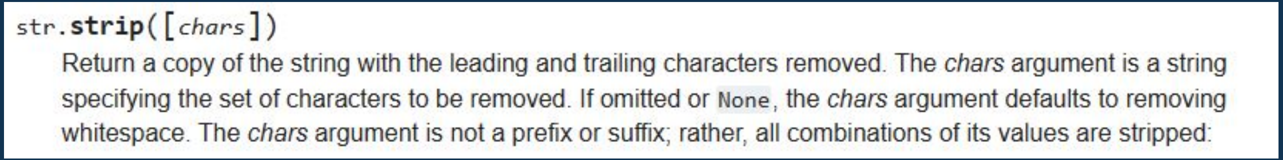
`str.join(iterable)`

Return a string which is the concatenation of the strings in *iterable*. A `TypeError` will be raised if there are any non-string values in *iterable*, including `bytes` objects. The separator between elements is the string providing this method.



`str.replace(old, new[, count])`

Return a copy of the string with all occurrences of substring *old* replaced by *new*. If the optional argument *count* is given, only the first *count* occurrences are replaced.



`str.strip([chars])`

Return a copy of the string with the leading and trailing characters removed. The *chars* argument is a string specifying the set of characters to be removed. If omitted or `None`, the *chars* argument defaults to removing whitespace. The *chars* argument is not a prefix or suffix; rather, all combinations of its values are stripped:

https://docs.python.org/3/library/stdtypes.html#string-methods

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 pow(), which returns x(the base) raised to the power of y (exponent) the value of a number to the power of something. There are two ways to access this:

  -
    ```
    import math
    my_result = math.pow(x,y)
    ```

  -
    ```
    from math import pow
    my_result =pow(x,y)
    ```

https://docs.python.org/3/library/math.html

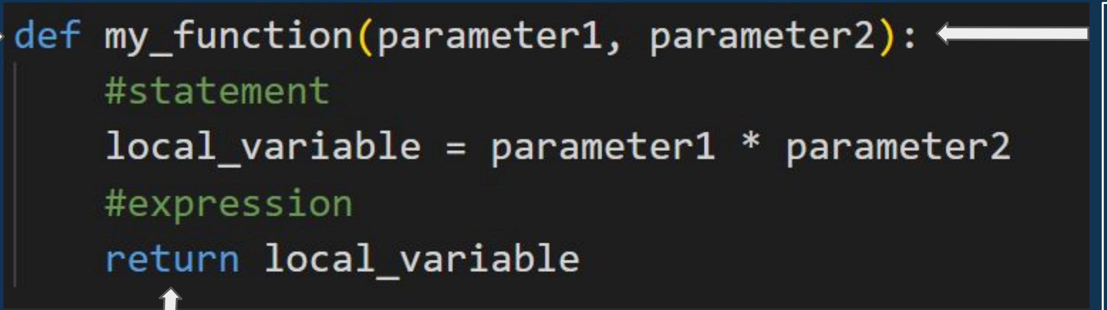
Some Important Terms

- **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.

Self-defined functions

- Reusable and Organised block of code.
- The general syntax of a function:

def-tells Python you are defining a function



```python
def my_function(parameter1, parameter2):
    #statement
    local_variable = parameter1 * parameter2
    #expression
    return local_variable
```

Parameters can take **required positional input** or **optional keyword** input (default values)

return - if your function returns a value, then use this keyword to return it.

Parameters - The defined input of a function.

Hyperiondev

Calling Functions

- Declare a variable to store the return value
- Give arguments to the parameters of the function

```
answer = my_function(1, 9)
```

Arguments - The values passed to parameters.

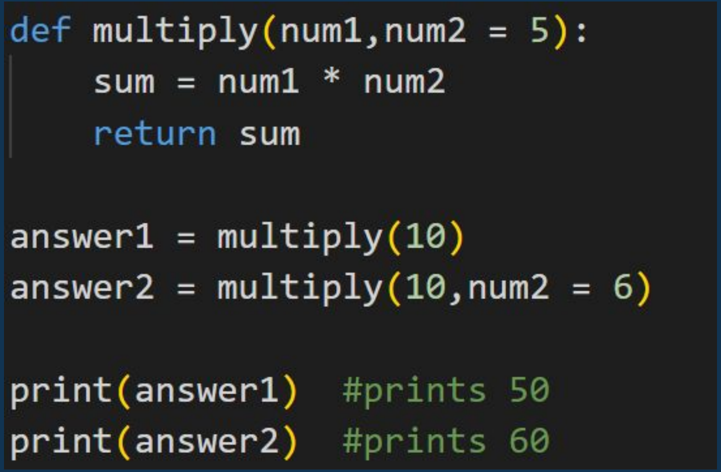
- To display the output of the function you need to call print on the variable.

```
# Print the output of the function for the 'answer' instance
print(answer)
```

Hyperiondev

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).

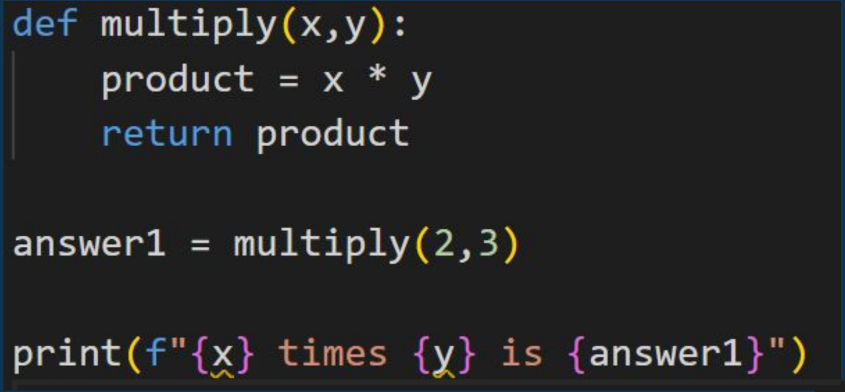


```
def multiply(num1,num2 = 5):
    sum = num1 * num2
    return sum

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

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

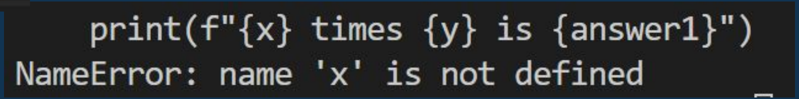
Hyperiondev

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.



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

answer1 = multiply(2,3)

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

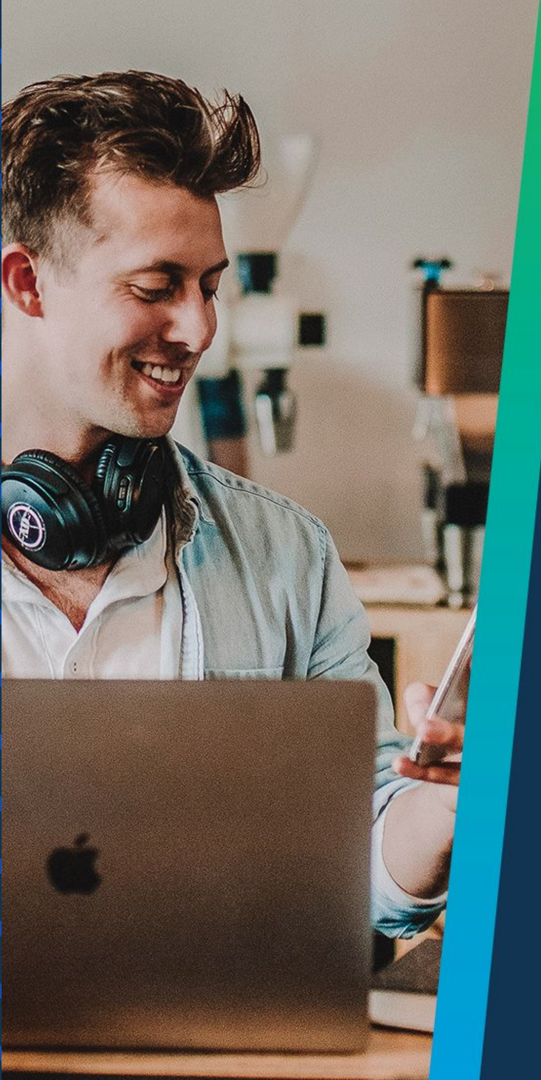


```
    print(f"{x} times {y} is {answer1}")
NameError: name 'x' is not defined
```

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