Python Programming Fundamentals

Programming Functions

Function is a group of related statements that perform a specific task.

Function named blocks of code that are designed to do one specific job.

Functions provide organized, reusable and modular code to perform a set of specific actions. Functions simplify the coding process, prevent redundant logic, and make the code easier to follow.

Functions help break our program into smaller and modular chunks. As our program grows larger and larger, functions make it more organized and manageable. Functions avoids repetition and makes code reusable.

A function is like a miniprogram within a program.

Function is a block of code that robotically does the same thing again and again, whenever you invoke its name. It saves you repetitive coding and makes your code easier to understand.

A major purpose of functions is to group code that gets executed multiple times. Without a function defined, you would have to copy and paste code each time.

Function is a piece of code written to carry out a specified task. To carry out that specific task, the function might or might not need multiple inputs. When the task is carried out, the function can or can not return one or more values.

Functions vs Methods

A **method** refers to a function which is part of a class. You access it with an instance or object of the class.

A **function** doesn't have this restriction, it just refers to a standalone function.

This means that `all methods are functions, but not all functions are methods.`

Python Functions / Methods

- Built-in
  - Functions
  - Methods
- User-Defined
  - Functions
  - Methods

- Main as a Function
- Anonymous Functions or Lambda Functions
- Generator Functions
- Special Functions
- Asynchronous Functions

Built-in Functions

Python built-in functions, such as `help()` to ask for help, `print()` to print an object etc.

In [4]:
```python
print('Python Built-in Functions')
```
```
Python Built-in Functions
```

In [5]:
```python
help(print)
```
```
Help on built-in function print in module builtins:

print(...)
    print(value, ..., sep=' ', end='\n', file=sys.stdout, flush=False)

    Prints the values to a stream, or to sys.stdout by default.
    Optional keyword arguments:
    file:  a file-like object (stream); defaults to the current sys.stdout.
    sep:   string inserted between values, default a space.
    end:   string appended after the last value, default a newline.
    flush: whether to forcibly flush the stream.
```

In [7]:
```python
#print(dir(__builtins__))
```

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

Built-in Functions Link: https://docs.python.org/3/library/functions.html

Built-in Methods

This is really a different disguise of a built-in function.

An example of a built-in method is `string.format()`, assuming string is a `str` object.

Built-in String Methods

- Below are some of String Methods:
  - upper
  - lower
  - swapcase
  - replace
  - count
  - capitalize
  - find
  - strip
  - startswith

- Method call is know as Invocation
- Multiple methods can be invoked in a single line
- Try the following code:
  - x = 'hello world'
  - x.startswith('h')
  - x.startswith('H')
  - x.capitalize().startswith('H')

Python Strings

In [15]:
```python
str?
```
```
Init signature: str(self, /, *args, **kwargs)
Docstring:
str(object='') -> str
str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or
errors is specified, then the object must expose a data buffer
that will be decoded using the given encoding and error handler.
Otherwise, returns the result of object.__str__() (if defined)
or repr(object).
encoding defaults to sys.getdefaultencoding().
errors defaults to 'strict'.
Type:           type
Subclasses:     DeferredConfigString, _rstr, LSString, include, ColorDepth, Keys, InputMode, CompleteStyle, Sor
tKey
```

In [18]:
```python
#help(str)
```

In [8]:
```python
string = "string built-in methods"
```

In [9]:
```python
print(type(string))
```
```
<class 'str'>
```

In [10]:
```python
# string.<press tab display all methods>
```

In [11]:
```python
string.title()
```
Out[11]:
```
'String Built-In Methods'
```

In [19]:
```python
str.capitalize?
```
```
Signature: str.capitalize(self, /)
Docstring:
Return a capitalized version of the string.

More specifically, make the first character have upper case and the rest lower
case.
Type:      method_descriptor
```

In [20]:
```python
lang = "python Programming"
```

In [21]:
```python
new_lang = lang.capitalize()
```

In [22]:
```python
new_lang
```
Out[22]:
```
'Python programming'
```

Python String Indexing, Slicing,

Indexing and Slicing and Other String Operations

- Indexing and Slicing are very common String Operations
- Each character in a string is identified by an index
- Index starts from 0 (not 1)
- Slicing is when we select multiple characters from a string
- m1 = 'hello', m1[0] will give us 'h'
- m2 = 'hello', m2[:2] gives us 'he'
- Try: m3 = 'abcdefghij', m3[2:8:2]

In [23]:
```python
# Indexing
# - Forward Index  --- 0 to N-1
# - Reverse Index  --- -1 to -N
```

In [24]:
```python
lang = "python"
```

In [25]:
```python
lang[-len(lang)]   # p
```
Out[25]:
```
'p'
```

In [26]:
```python
lang[-1]
```
Out[26]:
```
'n'
```

In [27]:
```python
len(lang)
```
Out[27]:
```
6
```

In [32]:
```python
# Accessing
```

In [33]:
```python
lang[len(lang)-1]   # n
```
Out[33]:
```
'n'
```

In [34]:
```python
lang[0]
```
Out[34]:
```
'p'
```

Slicing

In [39]:
```python
lang = "python"
```

In [40]:
```python
lang[1::2]
```
Out[40]:
```
'yhn'
```

In [41]:
```python
lang[::2]
```
Out[41]:
```
'pto'
```

In [42]:
```python
lang[::1]
```
Out[42]:
```
'python'
```

In [43]:
```python
lang[::-2]
```
Out[43]:
```
'nhy'
```

In [44]:
```python
lang[::]
```
Out[44]:
```
'python'
```

In [45]:
```python
lang[:]
```
Out[45]:
```
'python'
```

In [46]:
```python
lang[0:len(lang):]
```
Out[46]:
```
'python'
```

In [47]:
```python
lang[:5]
```
Out[47]:
```
'pytho'
```

In [48]:
```python
lang[1:len(lang)]   # 1:6
```
Out[48]:
```
'ython'
```

In [49]:
```python
lang = 'python dynamic programming'
```

In [50]:
```python
lang[::]
```
Out[50]:
```
'python dynamic programming'
```

In [51]:
```python
lang[:]
```
Out[51]:
```
'python dynamic programming'
```

In [52]:
```python
data = lang[:]
```

In [53]:
```python
data
```
Out[53]:
```
'python dynamic programming'
```

In [54]:
```python
lang[2:]
```
Out[54]:
```
'thon dynamic programming'
```

In [55]:
```python
lang[:-2]
```
Out[55]:
```
'python dynamic programmi'
```

String Find Method

In [58]:
```python
str.find?
```
```
Docstring:
S.find(sub[, start[, end]]) -> int

Return the lowest index in S where substring sub is found,
such that sub is contained within S[start:end].  Optional
arguments start and end are interpreted as in slice notation.

Return -1 on failure.
Type:      method_descriptor
```

The find() method returns the index of first occurrence of the substring (if found). If not found, it returns -1.

The syntax of find() method is:

`str.find(sub[, start[, end]])`

String find() Parameters()

- The find() method takes maximum of three parameters:
  - **sub** - It's the substring to be searched in the `str` string.
  - **start** and **end** (optional) - substring is searched within `str[start:end]`

Return Value from find()

- The find() method returns an integer value.
  - If substring exists inside the string, it returns the index of first occurence of the substring.
  - If substring doesn't exist inside the string, it returns -1.

Example 1: find() With No start and end Argument

In [62]:
```python
s1 = 'rizwan.datahub@gmail.com'
```

In [63]:
```python
print(s1.find('gmail'))
```
```
15
```

In [64]:
```python
print(s1.find('com'))
```
```
21
```

In [66]:
```python
print(s1.find('outlook'))
```
```
-1
```

In [67]:
```python
print(s1.find('outlook') == -1)
```
```
True
```

In [68]:
```python
print(s1.find(' '))
```
```
-1
```

Example 2: find() With start and end Arguments

In [70]:
```python
s1
```
Out[70]:
```
'rizwan.datahub@gmail.com'
```

In [71]:
```python
print(s1.find('@'))
```
```
14
```

In [72]:
```python
print(s1.find('@', 10))
```
```
14
```

In [73]:
```python
print(s1.find('@', 16))
```
```
-1
```

In [74]:
```python
print(s1.find('@', 10, 15))
```
```
14
```

In [75]:
```python
print(s1.find('@', 15, 20))
```
```
-1
```

Solution - String Parsing Problem

In [77]:
```python
m = 'From rizwan.datahub@gmail.com Wed February 24 20:05:37 2020'
```

Expected Output: `gmail.com`

In [78]:
```python
print(m[m.find('@')+1 : m.find(' ', m.find('@'))])
```
```
gmail.com
```

String Formatting

In [79]:
```python
str.format?
```
```
Docstring:
S.format(*args, **kwargs) -> str

Return a formatted version of S, using substitutions from args and kwargs.
The substitutions are identified by braces ('{' and '}').
Type:      method_descriptor
```

In [103…
```python
# x: int = 5
```

In [106…
```python
# name: str = 'xyz'
```

In [107…
```python
# data: list = [2, 6, 6]
```

'New Style' String Formatting (str.format)

In [108…
```python
x = 4
y = "xyz"
z = True
```

In [129…
```python
result = "1st Value {age}, 2nd Value {p2}, 3rd Value {p3}".format(age=x, p2=y, p3=z)
```

In [130…
```python
print(result)
```
```
1st Value 4, 2nd Value xyz, 3rd Value True
```

String Interpolation / f-Strings (Python 3.6+)

(f-string)

PEP 3101, PEP 498, PEP 501

In [154…
```python
x = 10.43224
y = "xyz"
z = True
```

In [159…
```python
# string interpolation
result = f"1st Value {x}, 2nd Value {y}, 3rd Value {z}"
```

In [160…
```python
print(result)
```
```
1st Value 10.43224, 2nd Value xyz, 3rd Value True
```

In [161…
```python
# +
# *
# in, not in
```

Concatenation

In [163…
```python
'xyz' + 'abc'
```
Out[163…
```
'xyzabc'
```

In [164…
```python
x = 10.43224
y = "xyz"
z = True
```

In [180…
```python
# True + True # 2
```

In [181…
```python
# issubclass(bool, int)
```

In [177…
```python
# 'xyz' + str(5)
```

In [165…
```python
print("1st Value: " + str(x) + ", 2nd Value: " +  y + ", 3rd Value: " + str(z))
```
```
1st Value: 10.43224, 2nd Value: xyz, 3rd Value: True
```

In [168…
```python
# 3 ** 3
```

In [169…
```python
data = "xyz "
```

In [170…
```python
data * 5
```
Out[170…
```
'xyz xyz xyz xyz xyz '
```

In [171…
```python
lang = "Python programming"
```

In [182…
```python
"pthon" not in lang.lower()
```
Out[182…
```
True
```

In [186…
```python
# from string import Template
```

In [185…
```python
# Template?
```

In [187…
```python
# count
# replace
# upper
# find
# lower
# index
# split
# strip
```

In [225…
```python
data = 'python-programmiong'
```

In [235…
```python
# data.index('oo', 10)
```

In [221…
```python
new_data = data.split('-')
```

In [222…
```python
new_data
```
Out[222…
```
['python', 'programmiong']
```

In [201…
```python
# type(new_data)
```

In [191…
```python
# type(data)
```

In [236…
```python
data = 'python-programmiong'
```

In [237…
```python
new_data = data.replace('-', '*')
```

In [238…
```python
new_data
```
Out[238…
```
'python*programmiong'
```

In [239…
```python
data
```
Out[239…
```
'python-programmiong'
```

In [ ]:
```python
# Problem
```

In [ ]:
```python
msg = "From ali.ahmad@google.com Tue 15 March 2022 12:06:00"
```

In [ ]:
```python
# msg.find('@')
```

In [ ]:
```python
msg[msg.index('@')+1: msg.index(" ", msg.index("@"))]
```
Out[ ]:
```
'google.com'
```

In [ ]:
```python
msg[msg.find("@")+1 :msg.find(" ", msg.index("@")) ]
```
Out[ ]:
```
'google.com'
```

In [ ]:
```python
msg.find
```

In [ ]:
```python
msg.split('@')[1].split()[0]
```
Out[ ]:
```
'corvit.com.pk'
```

Happy Learning 😉