Python Crash Course

Interospection

Introspection is an act of self examination. In computer programming, introspection is the ability to determine type or properties of objects at runtime.

To introspect you use `isinstance`

In [5]:
```
l = [2,3,4,5]
t = (6,7,8,9)
d = {2: "a",
     3: "b",
     4: "c",
     5: "d"}

print(isinstance(l, list))
print(isinstance(t, tuple))
print(isinstance(d, dict))
```

```
True
True
True
```

dir()

Now the dir() function return all the attributes of any Object
If no parameters are passed it returns a list of names in the current local scope.

In [6]:
```
dir(l)
```

Out[6]:
```
['__add__',
 '__class__',
 '__contains__',
 '__delattr__',
 '__delitem__',
 '__dir__',
 '__doc__',
 '__eq__',
 '__format__',
 '__ge__',
 '__getattribute__',
 '__getitem__',
 '__gt__',
 '__hash__',
 '__iadd__',
 '__imul__',
 '__init__',
 '__init_subclass__',
 '__iter__',
 '__le__',
 '__len__',
 '__lt__',
 '__mul__',
 '__ne__',
 '__new__',
 '__reduce__',
 '__reduce_ex__',
 '__repr__',
 '__reversed__',
 '__rmul__',
 '__setattr__',
 '__setitem__',
 '__sizeof__',
 '__str__',
 '__subclasshook__',
 'append',
 'clear',
 'copy',
 'count',
 'extend',
 'index',
 'insert',
 'pop',
 'remove',
 'reverse',
 'sort']
```

Type()

This return the type of Object

In [7]:
```
type(l)
```

Out[7]:
```
list
```

Help

The help function gives you a nic documentation on the object, its attributes , its types, its usage etc.

In [8]:
```
help(l)
```

```
Help on list object:

class list(object)
 |  list(iterable=(), /)
 |
 |  Built-in mutable sequence.
 |
 |  If no argument is given, the constructor creates a new empty list.
 |  The argument must be an iterable if specified.
 |
 |  Methods defined here:
 |
 |  __add__(self, value, /)
 |      Return self+value.
 |
 |  __contains__(self, key, /)
 |      Return key in self.
 |
 |  __delitem__(self, key, /)
 |      Delete self[key].
 |
 |  __eq__(self, value, /)
 |      Return self==value.
 |
 |  __ge__(self, value, /)
 |      Return self>=value.
 |
 |  __getattribute__(self, name, /)
 |      Return getattr(self, name).
 |
 |  __getitem__(...)
 |      x.__getitem__(y) <==> x[y]
 |
 |  __gt__(self, value, /)
 |      Return self>value.
 |
 |  __iadd__(self, value, /)
 |      Implement self+=value.
 |
 |  __imul__(self, value, /)
 |      Implement self*=value.
 |
 |  __init__(self, /, *args, **kwargs)
 |      Initialize self.  See help(type(self)) for accurate signature.
 |
 |  __iter__(self, /)
 |      Implement iter(self).
 |
 |  __le__(self, value, /)
 |      Return self<=value.
 |
 |  __len__(self, /)
 |      Return len(self).
 |
 |  __lt__(self, value, /)
```

```
|      Return self<value.
|
|  __mul__(self, value, /)
|      Return self*value.
|
|  __ne__(self, value, /)
|      Return self!=value.
|
|  __repr__(self, /)
|      Return repr(self).
|
|  __reversed__(self, /)
|      Return a reverse iterator over the list.
|
|  __rmul__(self, value, /)
|      Return value*self.
|
|  __setitem__(self, key, value, /)
|      Set self[key] to value.
|
|  __sizeof__(self, /)
|      Return the size of the list in memory, in bytes.
|
|  append(self, object, /)
|      Append object to the end of the list.
|
|  clear(self, /)
|      Remove all items from list.
|
|  copy(self, /)
|      Return a shallow copy of the list.
|
|  count(self, value, /)
|      Return number of occurrences of value.
|
|  extend(self, iterable, /)
|      Extend list by appending elements from the iterable.
|
|  index(self, value, start=0, stop=9223372036854775807, /)
|      Return first index of value.
|
|      Raises ValueError if the value is not present.
|
|  insert(self, index, object, /)
|      Insert object before index.
|
|  pop(self, index=-1, /)
|      Remove and return item at index (default last).
|
|      Raises IndexError if list is empty or index is out of range.
|
|  remove(self, value, /)
|      Remove first occurrence of value.
|
|      Raises ValueError if the value is not present.
|
|  reverse(self, /)
```

```
|      Reverse *IN PLACE*.
|
|  sort(self, /, *, key=None, reverse=False)
|      Stable sort *IN PLACE*.
|
|  ----------------------------------------------------------------------
|  Static methods defined here:
|
|  __new__(*args, **kwargs) from builtins.type
|      Create and return a new object.  See help(type) for accurate signature.
|
|  ----------------------------------------------------------------------
|  Data and other attributes defined here:
|
|  __hash__ = None
```

Try & Except block

** Error handling in Python is done through the use of exceptions that are caught in try blocks and handled in except blocks. **

For example lets put the below block in a try block to handle a condition of avoiding division by zero

In [9]:
```
7/0
```

```
---------------------------------------------------------------------------
ZeroDivisionError                         Traceback (most recent call last)
<ipython-input-9-ff4b999c47a3> in <module>
----> 1 7/0

ZeroDivisionError: division by zero
```

This is how we avoid zero Division

In [10]:
```
try:
    print(7 / 0)

except Exception as e:
    print("Exception: ",e)
```

```
Exception:  division by zero
```

A Simple Use case

In [14]:
```
a= int(input("Enter a number to divide 10 by: "))
try:
    result = 10 / a
    print(result)
except Exception as ZeroDivisionError:
    print("Undefined")
```

```
Enter a number to divide 10 by: 0
Undefined
```

Importing Python Modules/Packages/Libraries

There are multiple ways to import pacakges

You can Import Whole Packages and then use modules from those packages

In [15]:
```
import os
os.stat("Media/MP/apple.png")
```

Out[15]:
```
os.stat_result(st_mode=33206, st_ino=1125899907205628, st_dev=1254706318, st_nl
ink=1, st_uid=0, st_gid=0, st_size=12803, st_atime=1569236731, st_mtime=1538353
676, st_ctime=1569236731)
```

You can directly import modules from packages like this

In [ ]:
```
from os import stat

print(stat("Media/MP/apple.png"))
```

Star " * " imports are wildcard imports , they import all modules from the package and they can overide used names in your namespace.

In [16]:
```
from os import *

print(stat("Media/MP/apple.png"))
```

```
os.stat_result(st_mode=33206, st_ino=1125899907205628, st_dev=1254706318, st_nl
ink=1, st_uid=0, st_gid=0, st_size=12803, st_atime=1569236731, st_mtime=1538353
676, st_ctime=1569236731)
```

You can give any Alias to your Package name

In [17]:
```
import os as donkey

print(donkey.stat("Media/MP/apple.png"))
```

```
os.stat_result(st_mode=33206, st_ino=1125899907205628, st_dev=1254706318, st_nl
ink=1, st_uid=0, st_gid=0, st_size=12803, st_atime=1569236731, st_mtime=1538353
676, st_ctime=1569236731)
```

Object Oriented Programming

One of the popular approach to solve a programming problem is by creating objects. This is known as Object-Oriented Programming (OOP).

In [18]:
```
class Greeter:

    # Constructor
    def __init__(self, name, drink):
        self.name = name  # Create an instance variable
        self.drink = drink

    # method of class
    def greet(self):
            print ('Hello, {} likes {}'.format( self.name, self.drink))
```

Now lets Create the Constructor of the Class

In [19]:
```
g = Greeter('Taha','pepsi')  # Construct an instance of the Greeter class
```

Now lets use a method of this class

In [20]:
```
g.greet()
```

```
Hello, Taha likes pepsi
```

Similarly you can instert values at instantiation and use them in methods

In [21]:
```
m = Greeter('Mustafa','Juice')
m.greet()
g.greet()
```

```
Hello, Mustafa likes Juice
Hello, Taha likes pepsi
```

The END Folks