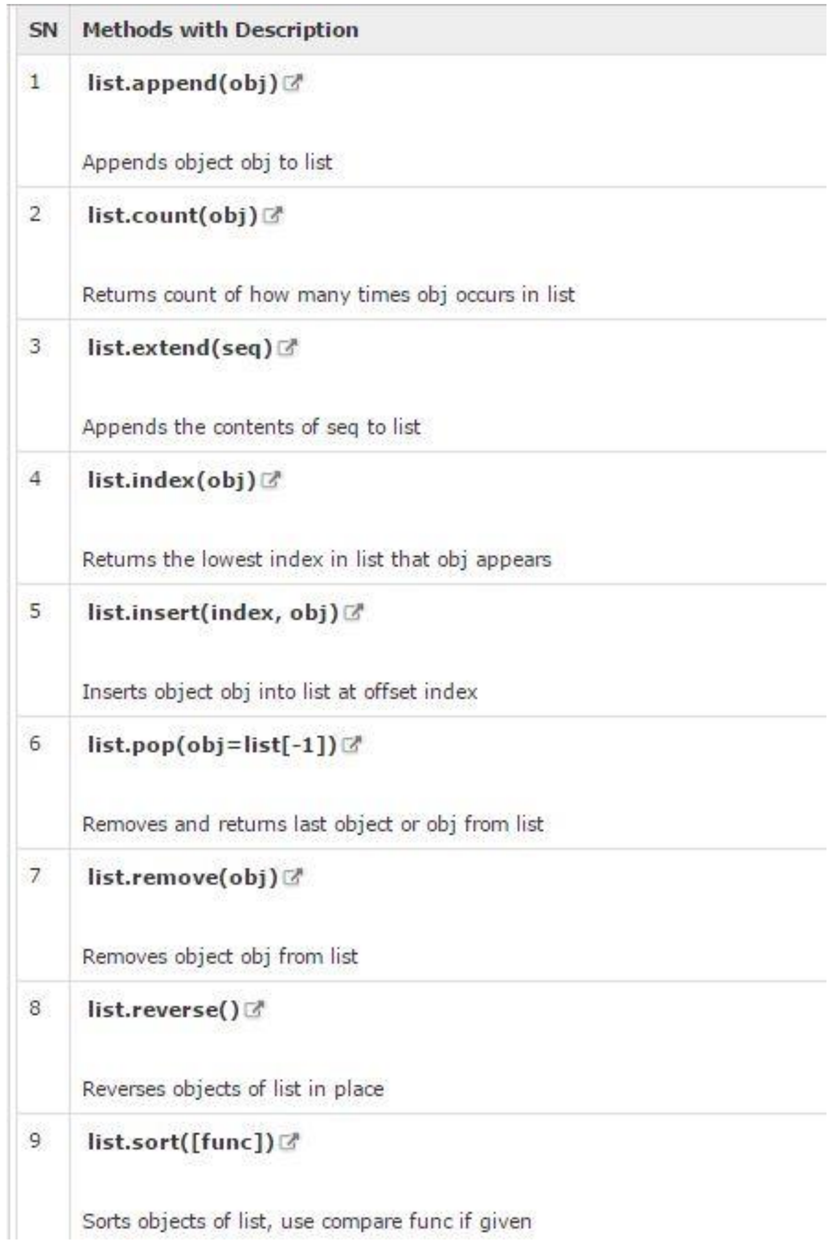
PYTHON

WEEK 8 – FUNCTIONS, FILE I/O, OOP-Intro

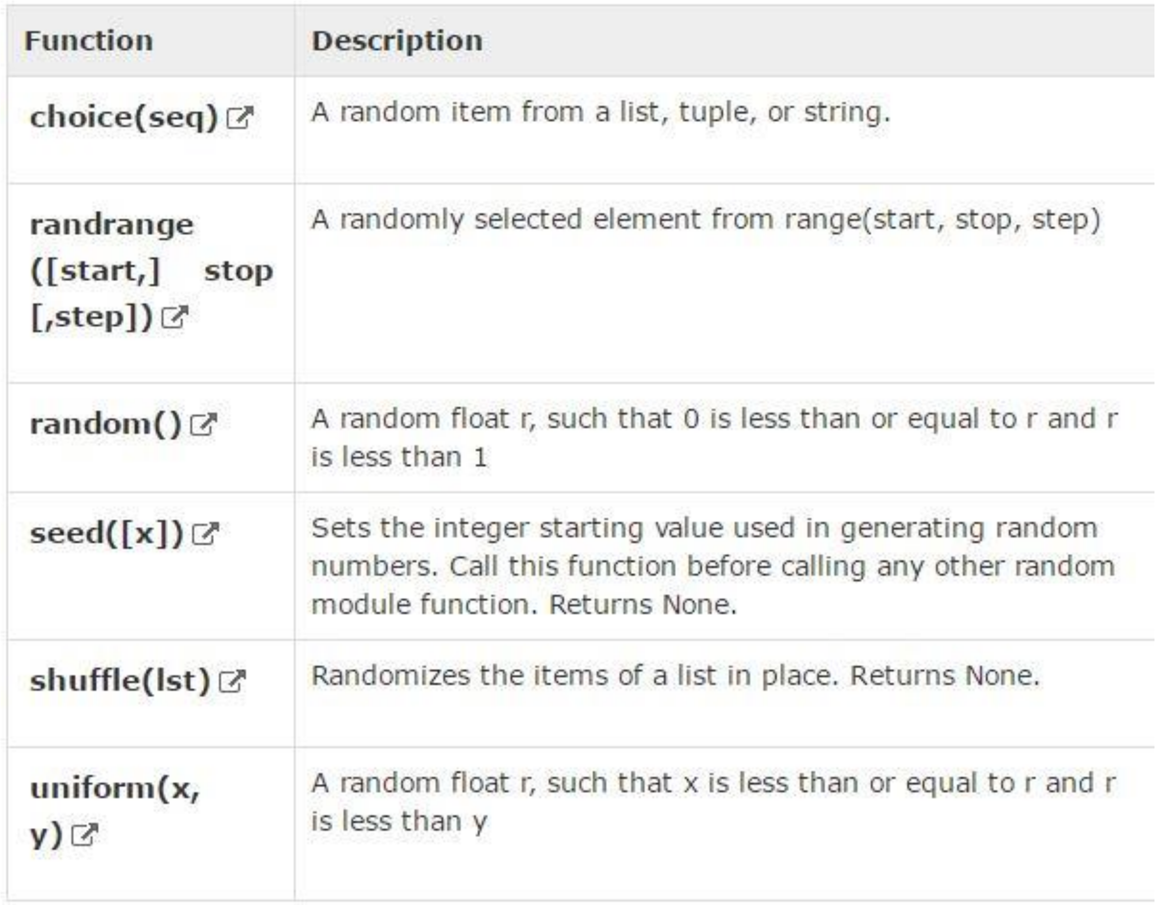
Academy of Innovative Education

Built-in List Functions & Methods II



SN	Methods with Description
1	**list.append(obj)** Appends object obj to list
2	**list.count(obj)** Returns count of how many times obj occurs in list
3	**list.extend(seq)** Appends the contents of seq to list
4	**list.index(obj)** Returns the lowest index in list that obj appears
5	**list.insert(index, obj)** Inserts object obj into list at offset index
6	**list.pop(obj=list[-1])** Removes and returns last object or obj from list
7	**list.remove(obj)** Removes object obj from list
8	**list.reverse()** Reverses objects of list in place
9	**list.sort([func])** Sorts objects of list, use compare func if given

Random Number Functions (*random* module)



Function	Description
choice(seq)	A random item from a list, tuple, or string.
randrange ([start,] stop [,step])	A randomly selected element from range(start, stop, step)
random()	A random float r, such that 0 is less than or equal to r and r is less than 1
seed([x])	Sets the integer starting value used in generating random numbers. Call this function before calling any other random module function. Returns None.
shuffle(lst)	Randomizes the items of a list in place. Returns None.
uniform(x, y)	A random float r, such that x is less than or equal to r and r is less than y

FUNCTIONS (1)

- A block of organized, reusable code that is used to perform a single, related action.
- Provide better modularity for your application and a high degree of code reusing.
- Defining a Function
  - begin with the keyword **def** followed by the function name and parentheses (( )).
  - Any input parameters or arguments should be placed within these parentheses. You can also define parameters inside these parentheses.
  - The code block within every function starts with a colon (:) and is indented.
  - The statement return [expression] exits a function, optionally passing back an expression to the caller.
- Calling a Function
  - Defining a function only gives it a name, specifies the parameters that are to be included in the function and structures the blocks of code.

NOTE: A return statement with no arguments is the same as return None.

FUNCTIONS (2)

- Pass by reference vs value
  - All parameters (arguments) in the Python language are passed by reference. It means if you change what a parameter refers to within a function, the change also reflects back in the calling function.

FUNCTIONS (3)

- Function Arguments

1. Required arguments
2. Keyword arguments
3. Default arguments
4. Variable-length arguments

Required arguments

- The arguments passed to a function in correct positional order.
- The number of arguments in the function call should match exactly with the function definition.

Keyword arguments

- Keyword arguments are related to the function calls. When you use keyword arguments in a function call, the caller identifies the arguments by the parameter name.
- This allows you to skip arguments or place them out of order because the Python interpreter is able to use the keywords provided to match the values with parameters.

Default arguments

- A default argument is an argument that assumes a default value if a value is not provided in the function call for that argument.

Variable-length arguments

- You may need to process a function for more arguments than you specified while defining the function.
- These arguments are called *variable-length* arguments and are not named in the function definition, unlike required and default arguments.
- An asterisk (*) is placed before the variable name that holds the values of all nonkeyword variable arguments. This tuple remains empty if no additional arguments are specified during the function call

The *Anonymous* Functions (1)

- These functions are called anonymous because they are not declared in the standard manner by using the *def* keyword.
- You can use the *lambda* keyword to create small anonymous functions.

The *Anonymous* Functions (2)

- Lambda
  - Lambda forms can take any number of arguments but return just one value in the form of an expression. They cannot contain commands or multiple expressions.
  - An anonymous function cannot be a direct call to print because lambda requires an expression
  - Lambda functions have their own local namespace and cannot access variables other than those in their parameter list and those in the global namespace.
  - Although it appears that lambda's are a one-line version of a function, they are not equivalent to inline statements in C or C++,

The *return* Statement

- The statement return [expression] exits a function, optionally passing back an expression to the caller.
- A return statement with no arguments is the same as return None.
- Multiple values can be returned, and it will be saved in a tuple

Scope of Variables

- All variables in a program may not be accessible at all locations in that program. This depends on where you have declared a variable.
- The scope of a variable determines the portion of the program where you can access a particular identifier. There are two basic scopes of variables in Python.

Global vs. Local variables

- Variables that are defined inside a function body have a local scope, and those defined outside have a global scope.
- This means that local variables can be accessed only inside the function in which they are declared, whereas global variables can be accessed throughout the program body by all functions.
- When you call a function, the variables declared inside it are brought into scope.

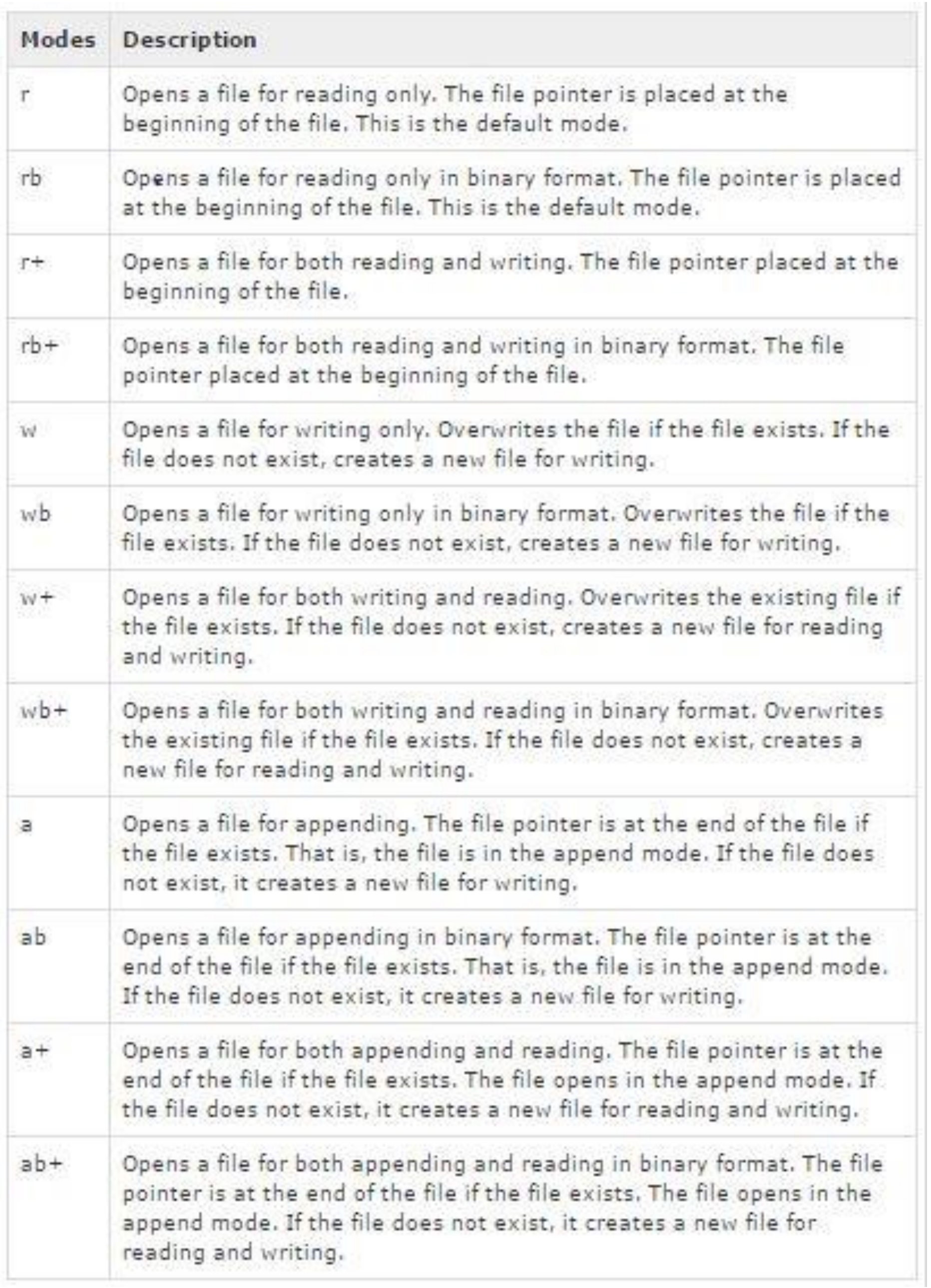
Files Input/output

- The *open* Function: This function creates a **file** object, which would be utilized to call other support methods associated with it.

Syntax:
```
file object = open(file_name [, access_mode][, buffering])
```

- **file_name:** The file_name argument is a string value that contains the name of the file that you want to access.
- **access_mode:** The access_mode determines the mode in which the file has to be opened, i.e., read, write, append, etc. A complete list of possible values is given below in the table. This is optional parameter and the default file access mode is read (r).
- **buffering:** If the buffering value is set to 0, no buffering takes place. If the buffering value is 1, line buffering is performed while accessing a file. If you specify the buffering value as an integer greater than 1, then buffering action is performed with the indicated buffer size. If negative, the buffer size is the system default(default behavior).

Modes of opening a file



Modes	Description
r	Opens a file for reading only. The file pointer is placed at the beginning of the file. This is the default mode.
rb	Opens a file for reading only in binary format. The file pointer is placed at the beginning of the file. This is the default mode.
r+	Opens a file for both reading and writing. The file pointer placed at the beginning of the file.
rb+	Opens a file for both reading and writing in binary format. The file pointer placed at the beginning of the file.
w	Opens a file for writing only. Overwrites the file if the file exists. If the file does not exist, creates a new file for writing.
wb	Opens a file for writing only in binary format. Overwrites the file if the file exists. If the file does not exist, creates a new file for writing.
w+	Opens a file for both writing and reading. Overwrites the existing file if the file exists. If the file does not exist, creates a new file for reading and writing.
wb+	Opens a file for both writing and reading in binary format. Overwrites the existing file if the file exists. If the file does not exist, creates a new file for reading and writing.
a	Opens a file for appending. The file pointer is at the end of the file if the file exists. That is, the file is in the append mode. If the file does not exist, it creates a new file for writing.
ab	Opens a file for appending in binary format. The file pointer is at the end of the file if the file exists. That is, the file is in the append mode. If the file does not exist, it creates a new file for writing.
a+	Opens a file for both appending and reading. The file pointer is at the end of the file if the file exists. The file opens in the append mode. If the file does not exist, it creates a new file for reading and writing.
ab+	Opens a file for both appending and reading in binary format. The file pointer is at the end of the file if the file exists. The file opens in the append mode. If the file does not exist, it creates a new file for reading and writing.

The *file* Object Attributes

- Once a file is opened and you have one *file* object, you can get various information related to that file.

Attribute	Description
file.closed	Returns true if file is closed, false otherwise.
file.mode	Returns access mode with which file was opened.
file.name	Returns name of the file.
file.softspace	Returns false if space explicitly required with print, true otherwise.

The *close()* Method

- The close() method of a *file* object flushes any unwritten information and closes the file object, after which no more writing can be done.

Syntax:

```
fileObject.close();
```

The *write()* Method

- The *write()* method writes any string to an open file. It is important to note that Python strings can have binary data and not just text.
- The write() method does not add a newline character ('\n') to the end of the string (you have to manually add separators)

Syntax:

```
fileObject.write(string);
```

The *read()* Method

- The *read()* method reads a string from an open file. It is important to note that Python strings can have binary data. apart from text data.

  Syntax:
```
fileObject.read([count]);
```
- Passed parameter is the number of bytes to be read from the opened file.
- This method starts reading from the beginning of the file and if *count* is missing, then it tries to read as much as possible, maybe until the end of file.
- readline(), readlines() methods are used to read a file in line by line

File Positions

- The *tell()* method tells you the current position within the file; in other words, the next read or write will occur at that many bytes from the beginning of the file.
- The *seek(offset[, from])* method changes the current file position. The *offset* argument indicates the number of bytes to be moved. The *from* argument specifies the reference position from where the bytes are to be moved.
    - If *from* is set to 0, it means use the beginning of the file as the reference position and 1 means use the current position as the reference position and if it is set to 2 then the end of the file would be taken as the reference position.

Renaming and Deleting Files

- The *rename()* method takes two arguments, the current filename and the new filename.

Syntax:
```
os.rename(current_file_name, new_file_name)
```

- The *remove()* method can be used to delete files by supplying the name of the file to be deleted as the argument.

Syntax:
```
os.remove(file_name)
```

NOTE: To as rename and delete files(file processing operations) in python, methods provided in **os** module have to be used

Directories in Python (1)

- *mkdir()* method of the **os** module to create directories in the current directory. You need to supply an argument to this method which contains the name of the directory to be created.

  Syntax:
  ```
  os.mkdir("newdir")
  ```
- *chdir()* method to change the current directory. The chdir() method takes an argument, which is the name of the directory that you want to make the current directory.

  Syntax:
  ```
  os.chdir("newdir")
  ```
- The *getcwd()* method displays the current working directory.

  Syntax:
  ```
  os.getcwd()
  ```

Directories in Python (2)

- The *rmdir()* method deletes the directory, which is passed as an argument in the method.

Syntax:

```
os.rmdir('dirname')
```

Python Object Oriented Programming

Overview of OOP Terminology

- **Class:** A user-defined prototype for an object that defines a set of attributes that characterize any object of the class. The attributes are data members (class variables and instance variables) and methods, accessed via dot notation.
- **Object:** A unique instance of a data structure that's defined by its class. An object comprises both data members (class variables and instance variables) and methods.
- **Instance:** An individual object of a certain class. An object obj that belongs to a class Circle, for example, is an instance of the class Circle.
- **Method :** A special kind of function that is defined in a class definition.
- **Class variable:** A variable that is shared by all instances of a class. Class variables are defined within a class but outside any of the class's methods. Class variables are not used as frequently as instance variables are.
- **Data member:** A class variable or instance variable that holds data associated with a class and its objects.
- **Instance variable:** A variable that is defined inside a method and belongs only to the current instance of a class.
- **Instance:** An individual object of a certain class. An object obj that belongs to a class Circle, for example, is an instance of the class Circle.

Creating a basic Class

- The *class* statement creates a new class definition. The name of the class immediately follows the keyword *class* followed by a colon

Syntax:

```
class ClassName:

      'Optional class documentation string'
      class_suite
```

- The class has a documentation string, which can be accessed via *ClassName.__doc__*.
- The *class_suite* consists of all the component statements defining class members, data attributes and functions.

Modifying Class Attributes

- You can add, remove, or modify attributes of classes and objects at any time. (No separations such as private, protected and public).
- Instead of using the normal statements to access attributes, you can use the following functions also.
  - The **getattr(obj, name[, default])** : to access the attribute of object.
  - The **hasattr(obj,name)** : to check if an attribute exists or not.
  - The **setattr(obj,name,value)** : to set an attribute. If attribute does not exist, then it would be created.
  - The **delattr(obj, name)** : to delete an attribute.

Built-In Class Attributes

- Every Python class keeps following built-in attributes and they can be accessed using dot operator like any other attribute.

  - **__dict__:** Dictionary containing the class's namespace.
  - **__doc__:** Class documentation string or none, if undefined.
  - **__name__:** Class name.
  - **__module__:** Module name in which the class is defined. This attribute is "__main__" in interactive mode.
  - **__bases__:** A possibly empty tuple containing the base classes, in the order of their occurrence in the base class list.