Cheat Sheet #2

Package:

A Python package usually consists of several modules. Physically, a package is a folder containing modules and maybe other folders that themselves may contain more folders and modules. Conceptually, it's a namespace. This simply means that a package's modules are bound together by a package name, by which they may be referenced. A package is a bundle of pre-built functionality that adds to the functionality available in base Python (a toolbox filled with tools).

Ex: Jupyter notebook, pandas, numpy

Object:

An object is simply a collection of data (variables) and methods (functions) that act on those data. Similarly, a class is a blueprint for that object.

Function:

A function is a block of organized, reusable code that is used to perform a single, related action. Functions provide better modularity for your application and a high degree of code reusing.

Method:

A method in python is somewhat similar to a function, except it is associated with object/classes. Methods in python are very similar to functions except for two major differences.

- The method is implicitly used for an object for which it is called.
- The method is accessible to data that is contained within the class.

Attribute:

Attributes of a class are function objects that define corresponding methods of its instances. They are used to implement access controls of the classes.
Attributes of a class can also be accessed using the following built-in methods and functions :

1. **getattr() –** This function is used to access the attribute of object.
2. **hasattr() –** This function is used to check if an attribute exist or not.
3. **setattr() –** This function is used to set an attribute. If the attribute does not exist, then it would be created.
4. **delattr() –** This function is used to delete an attribute. If you are accessing the attribute after deleting it raises error "class has no attribute".

Lists:

Lists are used to store multiple items in a single variable. They are one of 4 built-in data types in Python used to store collections of data. They are unique in the sense that the data stored in a list is ordered and changeable.

Lists are created using square brackets, quotations, and a name for the list.

For example:

List = [“Item1”, “Item2”, “Item3”]

Conditionals:

1. if

```
If (EXPRESSION == TRUE):
        Block of code
else:
        Block of code
```

2. if-else

```
If (EXPRESSION == TRUE):
        Statement (Body of the block)
else:
        Statement (Body of the block)
```

3. elif

```
if (condition):
      #Set of statement to execute if condition is true
elif (condition):
      #Set of statements to be executed when if condition is false and elif condition is true
else:
      #Set of statement to be executed when both if and elif #conditions are false
```

4. Nested if-else

```
if(condition):
      #Statements to execute if condition is true
if(condition):
      #Statements to execute if condition is true
      #end of nested if
      #end of if
```

For Loops:

The for loop is used to repeat a section of code known number of times or a sequence of operations that are performed over and over in some specified order.

Syntax:

```
for iterating_var in sequence:
    statements(s)
```

Example:

```
fruits = ["apple", "banana", "cherry"]
for x in fruits:
    print(x)
```

List Comprehensions:

List comprehension in Python is an easy and compact syntax for creating a list from a string or another list. It is a very concise way to create a new list by performing an operation on each item in the existing list. List comprehension is considerably faster than processing a list using the for loop.

Syntax:

[expression for element in iterable if condition]

Example without List Comprehension:

```
even_nums = []
for x in range(21):
    if x%2 == 0:
        even_nums.append(x)
print(even_nums)
```

Output: [0, 2, 4, 6, 8, 10, 12, 14, 16, 18, 20]

Example with List Comprehension:

```
even_nums = [x for x in range(21) if x%2 == 0]
print(even_nums)
```

Output: [0, 2, 4, 6, 8, 10, 12, 14, 16, 18, 20]