Assignment No. 6 (Python Basic - 1)

Submitted By: Sachin Dodake

Q.1. What are keywords in python? Using the keyword library, print all the python keywords.

`Ans-1`

> `Keywords:`
>
> - Python Keywords are special reserved words that convey a special meaning to the compiler/interpreter.
> - Each keyword has a special meaning and a specific operation.
> - Keywords in Python are reserved words that can not be used as a variable name, function name, or any other identifier.
> - The keywords only contains small letters except True, False, None keywords.

> - The various keyword can be found by using command **`help('keywords')`** as shown below

In [2]:

```python
help('keywords')
```

```
Here is a list of the Python keywords.  Enter any keyword to get more help.

False               class               from                or
None                continue            global              pass
True                def                 if                  raise
and                 del                 import              return
as                  elif                in                  try
assert              else                is                  while
async               except              lambda              with
await               finally             nonlocal            yield
break               for                 not
```

Q.2. What are the rules to create variables in python?

`Ans-2:`

> `Variables:`
>
> - Variable is a name that is used to refer to memory location i.e. variables are nothing but reserved memory locations to store values. This means that when you create a variable you reserve some space in memory.
> - Python variable is also known as an identifier and used to hold value.
> - In Python, we don't need to specify the type of variable because Python is a infer language and smart enough to get variable type
> - Based on the data type of a variable, the interpreter allocates memory and decides what can be stored in the reserved memory. Therefore, by assigning different data types to variables, you can store integers, decimals or characters in these variables.
> - It is recommended to use lowercase letters for the variable name. for eg. 'Sachin' and 'sachin' both are two different variables

> `Examples:`

```
                    i)  sachin = 10  (Integer type)
                    ii) a = 20.5  (float type)
                    iii) info  = "Data Science"   (string type)
                    iv) value = 10+20j   (Complex type)
                     v) digits = True    (boolean type)
                    vi)  x, y, z = 10, "Hello", "True" (assigning multiple values to
multiple variables)
```

> `Rules to define Variable:`
>
> - The first character of the variable must be an alphabet or underscore (_).
> - All the characters except the first character may be an alphabet of lower-case(a-z), upper-case (A-Z), underscore, or digit (0-9).
> - Identifier name must not contain any white-space, or special character (!, @, #, %, ^, &, *).
> - Identifier name must not be similar to any keyword defined in the language.
> - Identifier names are case sensitive; for example, my name, and MyName is not the same.

Q.3. What are the standards and conventions followed for the nomenclature of variables in python to improve code readability and maintainability?

`Ans-3`

> `Naming rules:`
>
> - A name only consists of characters from three groups: `digits (0-9), letters (a-z and A-Z), and underscores (_)`.
> - A name cannot start with a `digit`.
> - A name cannot coincide with one of Python’s `reserved words`. Reserved words or keywords are words that have special meaning to Python.

> `Naming convention:`
>
> - Use all `lowercase`. Ex: `name` instead of `Name`. One exception: class names should start with a capital letter and follow by lowercase letters.
> - Use `snake_case` convention (i.e., separate words by underscores, look like a snake). Ex: `gross_profit` instead of `grossProfit` or `GrossProfit`.
> - Should be `meaningful` and `easy` to remember. Ex: `interest_rate` instead of `r` or `ir`.
> - Should have a `reasonable length`. Ex: `sales_apr` instead of `sales_data_for_april`.
> - `Avoid` names of popular functions and modules. Ex: avoid `print, math, or collections`.

Q.4. What will happen if a `keyword` is used as a `variable name`?

`Ans-4`

> - `Keywords` are predefined, reserved words used in Python programming that have special meanings to the `compiler`.
> - We cannot use a `keyword` as a `variable name, function name, or any other identifier`. They are used to define the syntax and structure of the Python language.
> - `Keywords` define the `language’s syntax rules and structure`, and they cannot be used as `variable names`.

> - We cannot use keywords as variable names as they are reserved names that are built-in to Python.
> - The below code is wrong because we have used `continue` as a `variable name`. Therefore, it will show error as shown below,

In [10]:

```python
# explaination on que-4

continue = 'Python'       # it will throw ERROR

if = 1234                       # it will throw ERROR

import = True                  # it will throw ERROR
```

```
  Input In [10]
    continue = 'Python'       # it will throw ERROR
             ^
SyntaxError: invalid syntax
```

Q.5. For what purpose `def-keyword` is used?

`Ans-5`

> - The **`del`** keyword is used to delete objects.
> - In Python everything is an object, so the `del` keyword can also be used to delete variables, lists, or parts of a list etc.
> - **Syntax:**

```
      del obj_name              # used to delete `obj_name`
```

> **Example:**

In [1]:

```python
# explaination on del keyword

a = 11
b = 22

# we will get values of a & b
print(f"The value of 'b' before del is: {a}")             # this will print value of a
print(f"The value of 'b' before del is: {b}")             # this will print value of a


# deleting variable-b
del b                 # to delet variable b

# Let's try to print values of a & b
print(f"\nThe value of 'b' after del is: {a}")     # this will print value of a
print(f"The value of 'b' after del is: {b}")     # this will print give error
```

```
The value of 'b' before del is: 11
The value of 'b' before del is: 22

The value of 'b' after del is: 11
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Input In [1], in <cell line: 16>()
     14 # Let's try to print values of a & b
     15 print(f"\nThe value of 'b' after del is: {a}")     # this will print value of a
---> 16 print(f"The value of 'b' after del is: {b}")

NameError: name 'b' is not defined
```

Q.6. What is the operation of this `special character ‘\’`?

`Ans-6`

> - In Python strings, the **backslash ""** is a special character, also called the **`escape`** character.
> - In other words, it has a special meaning when we use it inside the strings. As the name suggests, the `escape character` escapes the characters in a string for a brief moment to introduce unique inclusion.
> - `Escape characters` or sequences are `illegal characters` for Python and `never get printed` as part of the output. When backslash is used in Python programming, it allows the program to escape the next characters.
> - **Syntax:**

```
      \Escape character
```

> - Types of Escape Sequence:

```
      i) \’   --------->  Single quotation
      ii) \\   --------->  Backslash
      iii) \n   --------->  New Line
      iv) \r   --------->  Carriage return
      v) \t   --------->  Tab
      vi) \b   --------->   Backspace
      vii) \f   --------->  Form feed
      viii) \ooo   --------->  Octal equivalent
      ix) \xhhh   --------->  Hexadecimal equivalent
```

Q.7. Give an example of the following conditions:

```
(i) Homogeneous list
(ii) Heterogeneous set
(iii) Homogeneous tuple
```

`Ans-7`

> `i) Homogeneous List:`
>
> - **List** is used to store the sequence of various types of data. A list can be defined as a collection of values or items of different types. Python lists are mutable type which implies that we may modify its element after it has been formed. The items in the list are separated with the comma (,) and enclosed with the square brackets [].
> - **Homogeneous List** The homogeneous list is the list which contains similar type of elements i.e. datatype of each and every elements should be same.
> - **Example:**

```
      i) list_age = [22, 54, 65, 78, 23, 53, 32, 44, 55]
      ii) list_laptop =  ['Dell', 'Lenovo', 'Acer', 'HP', 'Microsoft', 'Samsung']
```

> `ii) Heterogeneous Set`
>
> - **Set** is the collection of the unordered items. Each element in the set must be unique, immutable, and the sets remove the duplicate elements. Sets are mutable which means we can modify it after its creation.
> - **Heterogeneous Set** is a set which which contains all type of elements i.e. all types datatype.
> - **Example:**

```
      i) set_1 = {'Hello', 101, 2.5, 'Bye'}
      ii) set_2 = {3, 12.34, 'Sachin', True, }
```

> `iii) Homogeneous Tuple:`
>
> - **Tuple** is a group of items that are separated by commas. The indexing, nested objects, and repetitions of a tuple are somewhat like those of a list, however unlike a list, a tuple is immutable.
> - **Homogeneous Tuple** is a tuple which contains similar type of elements i.e. datatype of each and every elements should be same.
> - **Example:**

```
      i) tuple_city = ('Pune', 'Mumbai', 'Nashik', 'Delhi')
      ii) tuple_square = (1,4,9,16,26,36,49,64,81,100)
```

Q.8. Explain the mutable and immutable data types with proper explanation & examples.

`Ans-8:`

> - The `Mutable` or `Immutable` is the fancy word for explaining the property of data types of being able to get updated after being initialized. The basic explanation is thus: A mutable object is one whose internal state is changeable. On the contrary, once an immutable object in Python has been created, we cannot change it in any way.

> `i) Mutable Data Type:`
>
> - Anything is said to be mutable when anything can be modified or changed. The term `"mutable"` refers to an `object's capacity to modify its values`. These are frequently the things that hold a data collection.
> - Python mutable data types:

```
        a. Lists
        b. Dictionaries
        c. Sets
       d. User-Defined Classes (It depends on the user to define the characteristics of
the classes)
```

> - `Example-1:` Here, we created a list named m that contains 3 integers, 1, 2, and 3. After we change m by “popping” off the last value 3, the ID of m stays the same! So, objects of type int are immutable and objects of type list are mutable.

In [21]:

```python
# Example-1: Exaplination of "Mutable Data Type"

## assigning a integer value 43 to age variable

list_num = [1, 2, 3, 4, 5]        # defining a list with five element
print(list_num)                                    # printing list
print(id(list_num))                                # printing list

## removing last element of list
list_num.pop()                          # remove last element rom list
print(list_num)                                 # printing list
print(id(list_num))                             # printing list
```

```
[1, 2, 3, 4, 5]
1870661184768
[1, 2, 3, 4]
1870661184768
```

> `ii) Immutable Data Type:`
>
> - Anything is said to be immutable when anything can not be modified or changed. The term `"Immutable"` refers to `a state in which no change can occur over time`. A Python object is referred to as immutable if we cannot change its value over time. The value of these Python objects is fixed once they are made.
> - Python immutable data types:

```
        a. Numbers (Integer, Float, Complex, Decimal, Rational & Booleans)
        b. Tuples
        c. Strings
        d. Frozen Sets
```

> - `Example-2:` Let us try to asssigned a variable 'age' with two diferent integer value as show below.
> - We can see in example below, the value of age is not changed , it is still 42 is an integer number, of the type int, which is immutable. So, what happened is really that on the first line, age is a name that is set to point to an int object, whose value is 42.
> - When we type age = 43, what happens is that another object is created, of the type int and value 43 (also, the id will be different), and the name age is set to point to it. So, we didn’t change that 42 to 43. We actually just pointed age to a different location.
> - As you can see from printing id(age) before and after the second object named age was created, they are different.

In [15]:

```python
# Example-2: Exaplination of "Immutable Data Type"

## assigning a integer value 43 to age variable

age = 42                  # assigning 42 to `age` variable
print(id(age))                  # printing 'id' of variable
print(type(age))              # printing 'type' of variable
print(age)                          # printing 'value' of variable

## assigning a integer value 43 to age variable

age = 43                  # assigning 43 to `age` variable
print(id(age))                  # printing 'id' of variable
print(type(age))              # printing 'type' of variable
print(age)                          # printing 'value' of variable
```

```
1872453305872
<class 'int'>
42
1872453305904
<class 'int'>
43
```

Q.9. Write a code to create the given structure using only for loop.

```
    *
   ***
  *****
 *******
*********
```

In [39]:

```python
# Here, we have 5 rows pyramid

rows = 5

k = 0

for i in range(1, rows+1):
    for space in range(1, (rows-i)+1):
        print(end="  ")

    while k!=(2*i-1):
        print("* ", end="")
        k += 1

    k = 0
    print()
```

```
        *
      * * *
    * * * * *
  * * * * * * *
* * * * * * * * *
```

Q.10. Write a code to create the given structure using while loop.

```
|||||||||
  |||||||
   |||||
     |||
      |
```

```
In [50]: rows = 5+1

         for i in range(rows, 1, -1):
             for space in range(0, rows-i):
                 print("  ", end="")
             for j in range(i, 2*i-1):
                 print("| ", end="")
             for j in range(1, i-1):
                 print("| ", end="")
             print()
```

```
| | | | | | | | |
  | | | | | | |
    | | | | |
      | | |
        |
```