Identifiers

In [ ]:
```
Any name in python program is nothing but an identifier.
It can be varibale name , class name , function name etc.

Example:
    4x_y=10
    class Students name
    def add
```

Rules of Defining an identifiers:

In [ ]:
```
1.In identifier the allowed characters are :
        a. Alphabet Symbol(Either upper case or lower case)
        b. Digits(0,9)
        c. underscore(_)

    Example:  cash=10   #Valid
              ca$h=20  #Invalid

2.Identifier can not be started with a digit
    Example : hello123=200   #Valid
              123hello=300   #Invalid

3.Identifiers are case senstive(Even whole python is a case senstive)
    Example :  TOTAL=200 #This is one seperate variable
               total=300 #This is one seperate variable

4.Identifiers are not be used as a Keyword.
    Example: class=10   #Inavlid

5.In Python there is no any limit of identifier that means you can create any length identifer
  but it is recommended to use valid meaningful name.
    Example: Student_name="Rehan"  #Valid and Recommended
             xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx=200#Valid and Not Recommended

6.WIthin the identifier there must not be any space.
    Example: Student Name="Rehan" #Invalid
                Student_Name1="Rehan"  #Valid
```

Practice Questions on Identifiers

In [ ]:
```
Identify which of the following identifer are valid?

1.  123total = "Hello World"   #Invalid
2.  total123 = 10              #Valid
3.  ca$h     = 20.5            #Invalid
4.  def      = 99.90           #Invalid
5.  _abc_abc = "A"             #Valid
6.  if       = 98              #invalid
7.  java2share = 98            #Valid
```

Variables

In [ ]:
```
Here x is a variable that will hold the address of data 10 inside the memeory
x=10
identifier is used to identify the data(Its a name only).
variables are the name given to the memory location (Hold the value.)
Syntax:
    identifier_name=data
```

In [ ]:
```
Examples:
    student_name="Rehan"
    student_roll=56
    student_city=Hyderabad
```

Keywords

In [ ]:
```
In python some reversed/Predefined words are there and that words are used to give
specific functionality.Such type of words are known as Keywords
In python 3.10 we have 36 keywords are present.
```

In [8]:
```
import keyword
keyword.kwlist
```

Out[8]:
```
['False',
 'None',
 'True',
 '__peg_parser__',
 'and',
 'as',
 'assert',
 'async',
 'await',
 'break',
 'class',
 'continue',
 'def',
 'del',
 'elif',
 'else',
 'except',
 'finally',
 'for',
 'from',
 'global',
 'if',
 'import',
 'in',
 'is',
 'lambda',
 'nonlocal',
 'not',
 'or',
 'pass',
 'raise',
 'return',
 'try',
 'while',
 'with',
 'yield']
```

Everything is an Object

In [ ]:
```
Object --> The real world entity--> Behaviour and Attributes
Example: Dog:
         Behaviour(Functionality) --> Barking
         Attrbutites(Properties)--> 4 legs Two eyes
Example Mobile:
    Behaviour(Functionality)--> Calling/Camera
    Attributes(properties)-->Processor/Ram
Example Televsion :
    Behaviour(Functionality)-->Channel_Change/Volume
    Attributes(Properties)-->Size/Model/Rating
```

In [ ]:
```
In python also each and everything is an object.That is the reason whenever we are checking
the type we are getting answer as class xxx(datatype). That means each and every datatype
is an object in python.
```

In [17]:
```
x=10.5
print(type(x))
```

```
<class 'float'>
```

How are Objects Stored in Memory in Python

In [ ]:
```
Each and Everything in Python is an object. For storing an object in any programming
langugage we are having a private heap and In that private each and every object is stored.
```

Practice Questions on Number of Object

In [50]:
```
x=10.5 # 1 fLoat object

x=[] #1 list object

x={} #1 Dictionary object

x=[10,30,20]  # Total 4 objects 1 List object 3 Int object

x={10,20,30,10,20,30} #It is a set duplicates are not allowed so only 3 Int objects
                      #and 1 set object will be created

x=(10,20,30,40)  #Total 5 Object 1 Tuple object and 4 Int object
```

```
<class 'float'>
```

Mutability vs Immutability

In [ ]:
```
Mutability--> Content can be change
Immutability --> Content can not be changed
```

In [ ]:
```
Note --> All the standard datatype are immutable that means you can not perform any change in it.
if you are going to perform any change then because of that change a new object will be created
and the old variable is going to point that new object.

#All standard datatype are immutable --> int , float, string , boolean and complex
```

Immutability Check for Integer Datatype

In [53]:
```
x=10
print(id(x))
x=x+10
print(id(x))
```

```
1748904274512
1748904274832
```

Immutability Check for Float Datatype

In [54]:
```
x=10.5
print(id(x))
x=x+2.5
print(id(x))
```

```
1749020563120
1748991807056
```

Immutability Check for String Datatype

In [55]:
```
x="String1"
print(id(x))
x="String1"+"String2"
print(id(x))
```

```
1749020425712
1749020672688
```

Immutability Check for Complex Datatype

In [56]:
```
x=1+2j
print(id(x))
x=(1+2j)+(3+4j)
print(id(x))
```

```
1749020971888
1749020971184
```

Immutability Check for Boolean Datatype

In [57]:
```
#Boolean
x=True
print(id(x))
x=1+True
print(id(x))
```

```
140721025767528
1748904274256
```

Memory Utilization in Python

In [ ]:
```
Note: In python if there is any requirement of creating an object then python will never create
    the object immediately it will first check weather the object with the same
    content is already present in the heap memeory or not if the object with same content is
    already present then pvm will never create an object it will directly redirect the variable
    to the same object with same content
    These are applicable only for int  and string datatype.
```

In [59]:
```
#For Integer Datatype
x=10
y=10
print(id(x))
print(id(y))
```

```
1748904274512
1748904274512
```

In [60]:
```
#For Float Datatype
x=10.004
y=10.004
print(id(x))
print(id(y))
```

```
1749020565168
1749020971888
```

In [61]:
```
#For Complex Datatype
x=10+5j
y=10+5j
print(id(x))
print(id(y))
```

```
1749020972208
1749020972112
```

In [62]:
```
#For String Datatype
x="String1"
y="String1"
print(id(x))
print(id(y))
```

```
1749020993328
1749020993328
```

TypeCasting - Conversion from One Datatype to Another

In [ ]:
```
For Converting any Datatype to List we have --> list()
For Converting any Datatype to Tuple we have --> tuple()
For Converting any Datatype to Set we have --> set()
For Converting any Datatype to Dictionary we have --> dict()
```

Conversion of List to Tuple is Possible

In [67]:
```
x=[10,20,30,40,50]
y=tuple(x)
print(y)
```

```
(10, 20, 30, 40, 50)
```

Conversion of List to Set is Possible

In [68]:
```
x=[10,20,30,40,50]
y=set(x)
print(y)
```

```
{40, 10, 50, 20, 30}
```

Conversion of List to Dictionary is Not Possible

In [69]:
```
x=[10,20,30,40]
y=dict(x)
print(y)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Input In [69], in <cell line: 2>()
      1 x=[10,20,30,40]
----> 2 y=dict(x)
      3 print(y)

TypeError: cannot convert dictionary update sequence element #0 to a sequence
```

Conversion of Tuple to List is Possible

In [71]:
```
x=(10,20,30,40,50)
y=list(x)
print(y)
```

```
[10, 20, 30, 40, 50]
```

Conversion of Tuple to Set is Possible

In [72]:
```
x=(10,20,30,40)
y=set(x)
print(y)
```

```
{40, 10, 20, 30}
```

Conversion of Tuple to Dict is not Possible

In [74]:
```
x=(10,20,30,40)
y=dict(x)
print(y)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Input In [74], in <cell line: 2>()
      1 x=(10,20,30,40)
----> 2 y=dict(x)
      3 print(y)

TypeError: cannot convert dictionary update sequence element #0 to a sequence
```

Conversion of Set to List is Possible

In [75]:
```
x={10,20,30,40}
y=list(x)
print(y)
```

```
[40, 10, 20, 30]
```

Conversion of Set to Tuple is Possible

In [76]:
```
x={1,2,3}
y=tuple(x)
print(y)
```

```
(1, 2, 3)
```

Conversion of Set to Dictionary is Not Possible

In [77]:
```
x={1,3,5}
y=dict(x)
print(y)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Input In [77], in <cell line: 2>()
      1 x={1,3,5}
----> 2 y=dict(x)
      3 print(y)

TypeError: cannot convert dictionary update sequence element #0 to a sequence
```

Conversion of Dictionary to List is Possible

In [78]:
```
x={1:2,3:4,5:6}
y=list(x)
print(y)
```

```
[1, 3, 5]
```

Conversion of Dictionary to Tuple is Possible

In [79]:
```
x={1:2,3:4,5:6}
y=tuple(x)
print(y)
```

```
(1, 3, 5)
```

Conversion of Dictionary to Set is Possible

In [80]:
```
x={1:2,3:4,5:6}
y=set(x)
print(y)
```

```
{1, 3, 5}
```

Important Concepts

In [ ]:
```
In which of the follwing indexing is not important?
list --> Yes
set--> No
dictionary --> No
Tuple --> Yes
String -->Yes
```

In [ ]:
```
datatypes which can be use as a key in dictionary?

1.Tuple --> as  a key
2.float --> as a key
3.String --> as a key
4.Int --> as a key
5.Boolean --> as a key possible

datatypes which cannot be use as a key in dictionary?

list
set

Dictionary keys are unique.
```

In [ ]: