RULES FOR VARIABLES

In [26]:
```
# 1. Variables are case sensitive. eg : a, A, b, B etc
```

In [36]:
```
# 2. Variables cannot start with a number like 1a, 2a., but can contain numbers after first letter. eg: A1, Aam
```

In [38]:
```
# 3. Variables cannot contain special characters like !@#$%, except underscore '_'
```

In [ ]:
```
# 4. keywords from keyword.kwlist cannot be used. eg : for, if, or, as, class etc
```

VALUES

In [43]:
```
a = 5 #where a is known as variable and 5 is the value
type(a) # returns the type of value. in this case, it is "int"
```

Out[43]:
```
int
```

In [45]:
```
b = 8.0
type(b) # this type of value is called "float"
```

Out[45]:
```
float
```

In [47]:
```
c = "string"
type(c) # this type of value is called "string" it is a text that is enclosed in single quotes, double quotes o
```

Out[47]:
```
str
```

In [49]:
```
c = '''str
ing''' # triple single quotes can be used for multi-line strings. result is included with /
```

In [51]:
```
c
```

Out[51]:
```
'str\ning'
```

In [53]:
```
d = 50+0j # this type of value is called "complex", where is '50' real number, '0' is imaginery number and j is
```

In [55]:
```
e = True # this type of value is called "bool". it means true or false
type(e)
```

Out[55]:
```
bool
```

TYPE CASTING

In [65]:
```
# type casting means converting one type of value to another
```

In [67]:
```
a = 5 # this is an int value
```

In [69]:
```
a_float = float(a) # by using this syntax, we can convert int to float
```

In [71]:
```
type(a_float)
```

Out[71]:
```
float
```

In [75]:
```
a_str = str(a) # this syntax is used to convert int to str
```

In [77]:
```
type(a_str)
```

Out[77]:
```
str
```

In [81]:
```
a_bool = bool(a)
type(a_bool) # this syntax is used to convert int to bool
```

Out[81]:
```
bool
```

In [83]:
```
a_complex = complex(a)
type(a_complex) # this syntax is used to convert int to complex
```

Out[83]:
```
complex
```

ALLOWED CONVERSIONS :

In [90]:
```
# int --> float
# int --> str
# int --> bool
# int --> complex
# float --> int
# float --> str
# float --> bool
# float --> complex
# bool --> int
# bool --> float
# bool --> str
# bool --> complex
# str --> int, if str is a number and not text
# str --> float, if str is a number and not text
# str --> bool
# str --> complex, if str is a number and not text
# complex --> str
# complex --> bool
```

INDEXING

In [93]:
```
# indexing is used to return specific element from a string
```

In [42]:
```
Name = 'AamirBinRaheem'
```

In [44]:
```
# there are two types of indexing, forward and backward. forward indexing starts with '0' and goes from left to
# backward indexing starts with -1 and goes from right to left. it is inclusive
```

In [46]:
```
Name[1] #syntax for indexing always includes square brackets
```

Out[46]:
```
'a'
```

In [48]:
```
Name[-10]
```

Out[48]:
```
'r'
```

SLICING

In [51]:
```
#slicing is used to return a portion of a string
```

In [53]:
```
Name[0:10]
# the 10th letter 'h' is not included due to exclusivity
```

Out[53]:
```
'AamirBinRa'
```

In [55]:
```
Name[-5:-1] # all letters are included due to inclusivity
```

Out[55]:
```
'ahee'
```

In [57]:
```
Name[:2] # returns first 2 elements
```

Out[57]:
```
'Aa'
```

In [59]:
```
Name[-3:] # returns last 3 elements
```

Out[59]:
```
'eem'
```

In [61]:
```
Name[:] # returns all the elements of the string
```

Out[61]:
```
'AamirBinRaheem'
```

DATA STRUCTURE

In [3]:
```
# Data structure is a collection of data types. eg: (a = 1, 1.1, "ABCD", True)
```

In [5]:
```
# There are two types of data structures.
# Inbuild data structure - list, tuple, set, dict
# user defined data structure - stack, queue, tree, linked list
```

INBUILD DATA STRUCTURE - LIST

In [10]:
```
l = [] # List starts with the letter 'l' and square brackets
```

In [10]:
```
l = [] # List starts with the letter "l" and square brackets.
```

In [12]:
```
type(l)
```

Out[12]:
```
list
```

In [14]:
```
#list has many functions
```

In [ ]:
```
l.append() # adds an element at the last of the list
l.copy # copy one list to another
l.remove() # removes one element based on first occurrence
l.clear() # removes all the elements from the list
l.count() # counts the number of times an element is present in the list
l.pop() # removes an element and returns it. have to specify the index
l.append([]) # creates nested list
l.insert(a, b) # inserts an element before the index. where 'a' is the index, and 'b' is the element to be inse
l.extend() # adds the elements of one list to another list
l.index() # returns the index of the given element
l.sort(reverse=True) # sorts the list in descending order
l.sort(reverse=False) # sorts the list in ascending order
del l[a] # deletes the element based on indexing, where 'a' is the index
del l # deletes the list
l1 = l.copy # address of both 'l' and 'l1' will be the same
id(l) # returns the address of 'l'
l[index] = element # adds the given element at the given index
l0 = l1 + l2 # joins two lists to create a new list
1 in l1 # checks if '1' exists in the list, true if yes, false if no. This is called list membership
```

LIST SLICING

In [ ]:
```
# List slicing is used to return a portion of the list
l[0:3] # returns the elements between 0 and 3rd index
l[:3] # returns first three elements
l[-3:] # returns last 3 elements
l[::-1] # reverses the list
l[1:100:10] # returns the element at every 11th index, upto 100 indexes
l[-1][0] # if -1 is a string, returns the 0 element of that string. This is called nested slicing
```

STRIP

In [ ]:
```
a.lstrip() # removes leading space i.e left most space
a.rstrip() # removes trailing space i.e right most space
a.strip() # removes both leading and trailing spaces
```

ESCAPE CHARACTER

In [ ]:
```
# escape character i.e \ \ can be used to add quotations in a string.
# a = "Aamir Bin \"Raheem\""
# result : Aamir Bin "Raheem"
```

ENUMERATE

In [ ]:
```
for i in enumerate(l):
    print(i) # returns the indexes and their respective values
```

ALL/ANY

In [ ]:
```
any(l1) # if 0 is present/absent in the list, returns True
all(l1) # if 0 is present in the list, returns False
```

TUPLE

In [ ]:
```
# Tuple is just like a list, but it is denoted with () and is immutable.
t = () # tuple syntax
#tuple has only 2 functions
t.count() # counts the number of times an element repeats
t.index() # returns the index of the element
```

SET

In [ ]:
```
s = {}, s = set() # both syntaxes can be used to create a set
#set automatically arranges elements in ascending order
# duplicates are not allowed
# do not allow nested lists, tuples
# indexing or slicing not allowed
# set has many functions
# for loop allowed
for i in s:
    print(i)
# enumerate is also allowed
for i in enumerate(s):
    print(i)
# list membership allowed
5 in s, 4 in s etc
s.add() # adds an element
s.clear() # clears the entire set
s.remove() # removes an element
s.discard() # removes an element if present in the set, if not, then won't remove and won't give error
s.pop() # removes a random element and returns it. no arguments are allowed since set does not allowed indexing
s.update() # adds elements of another set to the given set
```

SET OPERATIONS

In [ ]:
```
s1.union(s2) OR s1 | s2 # returns unique elements from both the sets. different from "update" function
s1.intersection(s2) OR s1 & s2 # returns common element from both the sets
s1.difference(s2) OR s1 - s2 # in s1, returns the removes the common elements from s2 and returns rest of the e
s1.symmetric_difference(s2) OR s1 ^ s2 # removes the common elements and prints the rest from both the sets
s1.issubset(s2) # checks whether the elements from s1 exist in s2. True if yes, False if no
s1.issuperset(s2) # checks whether the elements from s1 exist in s2. True if yes, False if no
s1.isdisjoint(s2) # checks whether the elements from s1 do not exist in s2. True if yes, False if no
```

DICT

In [ ]:
```
# dict syntax - keys : values
#dict is mutable
d1 = {} # this is empty dict
d1 = {1 : "two", 2 : "two"} # this is how a dict looks like
print(d1[1]) # returns the value for key 1(in this case, it is "two")
d1.values() # returns all the values in a dict
d1.keys() # returns all the keys in a dict
d1.update(d2) #merges d1 and d2
d2 = d1.copy() # copies one dict to another
keys = {'a', 'b', 'c'}
value = [10, 20, 30]
d3 = dict.fromkeys(keys, value)
d3 # assigns each key, 3 given values
```

NUMPY

In [4]:
```
import numpy as np # syntax for importing numpy
np.__version__ # syntax for checking numpy version
```

Out[4]:
```
'1.26.4'
```

In [6]:
```
my_list = [0,1,2,3,4,5]
arr = np.array(my_list) #syntax for converting a list into array
```

In [8]:
```
arr
```

Out[8]:
```
array([0, 1, 2, 3, 4, 5])
```

In [10]:
```
type(arr) # type of array always returns as numpy.ndarray
```

Out[10]:
```
numpy.ndarray
```

In [28]:
```
np.arange(10) # prints elements from 0 to 9th index
```

Out[28]:
```
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

In [30]:
```
np.arange(3, 11) # prints elements from 3rd index to 10th index
```

Out[30]:
```
array([ 3,  4,  5,  6,  7,  8,  9, 10])
```

In [32]:
```
np.arange(3, 8, dtype = float) # prints elements from 3rd index to 8th index in float datatype
```

Out[32]:
```
array([3., 4., 5., 6., 7.])
```

In [34]:
```
np.zeros(10) # prints 9 zeros. default data type is float
```

Out[34]:
```
array([0., 0., 0., 0., 0., 0., 0., 0., 0., 0.])
```

In [36]:
```
np.zeros(10, dtype = int) # prints 9 zeros in int datatype
```

Out[36]:
```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0])
```

In [49]:
```
np.zeros((2,2)) # prints 10 rows and 10 columns
```

Out[49]:
```
array([[0., 0.],
       [0., 0.]])
```

In [51]:
```
np.zeros((2,2), dtype=int) # prints 10 rows and 10 columns in int datatype
```

Out[51]:
```
array([[0, 0],
       [0, 0]])
```

In [ ]:
```
np.random.rand(10) # generates 9 random float values
np.random.rand(2, 3) # generates 2 rows, 3 columns of float values
np.arange(0, 10).reshape(2, 5) # generates numbers from 0 to 9, in 2 rows and 5 columns. (the multiplication of
np.random.randint(10,40(10,10)) # generates 10 rows and 10 columns consisting of numbers between 10 and 39
```

MATRIX SLICING

In [1]:
```
import numpy as np
b = np.random.randint(10,40,(5,4))
```

In [ ]:
```
b[:] # returns the entire matrix
b[1,3] # returns the element at 1st row, 3rd column
b[0:2] # returns 0,1 rows
b[1:3] # returns 1,2 rows
b[0:-3] # returns rows between 0 to -2
b[::-1] # reverses the matrix
b[:, 1] # prints column 1
b[1] # returns row 1
b[2:6, 2:4] # returns the elements present between row 2 - 5, to column 2 - 3.
```

NUMPY OPERATIONS

In [ ]:
```
arr.max() # prints the maximum value
arr.min() # prints the minimum value
arr.mean() # prints the avg
from numpy import *
a = array([1,2,3,4,5,6,7,8,9,0])
median(a) # from numpy import * is used to import all the functions from numpy, and we can use this syntax for
```

NUMPY MASKING

In [ ]:
```
#numpy masking refers to apllying filters to a matrix to return speific elements.
arr<5 # compares the entire matrix with 5. if less than 5, true. if more than 5 false
arr[arr<5] # returns only those elements which are less than 5
```

METHOD OF IMPORTING IMAGE

In [ ]:
```
from PIL import Image
sigma = Image.open(r'image location in single quotes')
sigma
```