Variables

A variable is a location in memory used to store some data (value).

They are given unique names to differentiate between different memory locations. The rules for writing a variable name is same as the rules for writing identifiers in Python.

We don't need to declare a variable before using it. In Python, we simply assign a value to a variable and it will exist. We don't even have to declare the type of the variable. This is handled internally according to the type of value we assign to the variable.

Variable Assignments

In [1]:

```
#We use the assignment operator (=) to assign values to a variable

a = 10
b = 5.5
c = "ML"
```

Multiple Assignments

In [2]:

```
a, b, c = 10, 5.5, "ML"
```

In [3]:

```
a = b = c = "AI" #assign the same value to multiple variables at once
```

Storage Locations

In [4]:

```
x = 3

print(id(x))                #print address of variable x
```

```
140722081441632
```

In [5]:

```
y = 3

print(id(y))                #print address of variable y
```

```
140722081441632
```

Observation:

x and y points to same memory location

In [6]:

```
y = 2
print(id(y))                #print address of variable y
```

```
140722081441600
```

Data Types

Every value in Python has a datatype. Since everything is an object in Python programming, data types are actually classes and variables are instance (object) of these classes.

Numbers

Integers, floating point numbers and complex numbers falls under Python numbers category. They are defined as int, float and complex class in Python.

We can use the type() function to know which class a variable or a value belongs to and the isinstance() function to check if an object belongs to a particular class.

In [7]:

```
a = 5                                  #data type is implicitly set to integer
print(a, " is of type", type(a))
```

```
5  is of type <class 'int'>
```

In [8]:

```
a = 2.5                               #data type is changed to float
print(a, " is of type", type(a))
```

```
2.5  is of type <class 'float'>
```

In [9]:

```
a = 1 + 2j                             #data type is changed to complex number
print(a, " is complex number?")
print(isinstance(1+2j, complex))
```

```
(1+2j)  is complex number?
True
```

Boolean

Boolean represents the truth values False and True

In [10]:

```
a = True                             #a is a boolean type
print(type(a))
```

```
<class 'bool'>
```

Python Strings

String is sequence of Unicode characters.

We can use single quotes or double quotes to represent strings.

Multi-line strings can be denoted using triple quotes, ''' or """.

A string in Python consists of a series or sequence of characters - letters, numbers, and special characters.

Strings can be indexed - often synonymously called subscripted as well.

Similar to C, the first character of a string has the index 0.

In [12]:

```
s = "Namaste World!"
print(s)
```

```
Namaste World!
```

In [13]:

```
print(s[0])
#last char s[len(s)-1] or s[-1]
```

```
N
```

In [14]:

```
#slicing
s[5:]
```

Out[14]:

```
'te World!'
```

Python List

List is an ordered sequence of items. It is one of the most used datatype in Python and is very flexible. All the items in a list do not need to be of the same type.

Declaring a list is , Items separated by commas are enclosed within brackets [ ].

In [15]:

```
a = [10, 20.5, "Hello"]
print(a[1])               #print 1st index element
```

```
20.5
```

Lists are mutable, meaning, value of elements of a list can be altered.

In [16]:

```
a[1] = 30.7
print(a)
```

```
[10, 30.7, 'Hello']
```

Python Tuple

Tuple is an ordered sequence of items same as list.The only difference is that tuples are immutable. Tuples once created cannot be modified.

In [17]:

```
t = (1, 1.5, "ML")
```

In [18]:

```
print(t[1]) #extract particular element
```

```
1.5
```

In [19]:

```
t[1] = 1.25
```

```
---------------------------------------------------------------
-----------
TypeError                                 Traceback (most recent
call last)
<ipython-input-19-826a3e5c5c91> in <module>
----> 1 t[1] = 1.25

TypeError: 'tuple' object does not support item assignment
```

Python Set

Set is an unordered collection of unique items. Set is defined by values separated by comma inside braces { }. Items in a set are not ordered.

In [20]:

```
a = {10, 30, 20, 40, 5}
print(a)
```

```
{5, 40, 10, 20, 30}
```

In [21]:

```
print(type(a))            #print type of a
```

```
<class 'set'>
```

We can perform set operations like union, intersection on two sets. Set have unique values.

In [22]:

```
s = {10, 20, 20, 30, 30, 30}
print(s)                    #automatically set won't consider duplicate element
```

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

In [23]:

```
print(s[1]) #we can't print particular element in set because
            #it's unorder collections of items
```

```
---------------------------------------------------------------
-----------
TypeError                                 Traceback (most recent
call last)
<ipython-input-23-3e2f312e6983> in <module>
----> 1 print(s[1]) #we can't print particular element in set be
cause
      2             #it's unorder collections of items

TypeError: 'set' object is not subscriptable
```

Python Dictionary

Dictionary is an unordered collection of key-value pairs.

In Python, dictionaries are defined within braces {} with each item being a pair in the form key:value. Key and value can be of any type.

In [25]:

```
d = {'a': "apple", 'b': "bat"}
print(d['a'])
```

```
apple
```

Conversion between Datatypes

We can convert between different data types by using different type conversion functions like int(), float(), str() etc.

In [26]:

```
float(5)     #convert interger to float using float() method
```

Out[26]:

```
5.0
```

In [27]:

```
int(100.5)   #convert float to integer using int() method
```

Out[27]:

```
100
```

In [28]:

```
str(20)      #convert integer to string
```

Out[28]:

```
'20'
```

Conversion to and from string must contain compatible values.

In [29]:

```
int('10p')
```

```
---------------------------------------------------------------
-----------
ValueError                                Traceback (most recent
call last)
<ipython-input-29-8bb83f613c14> in <module>
----> 1 int('10p')

ValueError: invalid literal for int() with base 10: '10p'
```

In [31]:

```
user = "ruhi"
lines = 100

print("Congratulations, " + user + "! You just wrote " + str(lines) + " lines o
#remove str and gives error
```

```
Congratulations, ruhi! You just wrote 100 lines of code
```

We can convert one sequence to other

In [32]:

```
a = [1, 2, 3]

print(type(a))      #type of a is list

s = set(a)          #convert list to set using set() method

print(type(s))      #now type of s is set
```

```
<class 'list'>
<class 'set'>
```

In [33]:

```
list("Hello")       #convert String to list using list() method
```

Out[33]:

```
['H', 'e', 'l', 'l', 'o']
```

Python Comments

Comments are lines that exist in computer programs that are ignored by compilers and interpreters.

Including comments in programs makes code more readable for humans as it provides some information or explanation about what each part of a program is doing.

In general, it is a good idea to write comments while you are writing or updating a program as it is easy to forget your thought process later on, and comments written later may be less useful in the long term.

In Python, we use the hash (#) symbol to start writing a comment.

In [34]:

```
#Print Hello, world to console
print("Hello, world")
```

```
Hello, world
```

Multi Line Comments

If we have comments that extend multiple lines, one way of doing it is to use hash (#) in the beginning of each line.

In [35]:

```
#This is a long comment
#and it extends
#Multiple lines
```

Another way of doing this is to use triple quotes, either ''' or """.

In [36]:

```
"""This is also a
perfect example of
multi-line comments"""
```

Out[36]:

```
'This is also a\nperfect example of\nmulti-line comments'
```

DocString in python

Docstring is short for documentation string.

It is a string that occurs as the first statement in a module, function, class, or method definition.

In [39]:

```
def doubleNum(num):
    """
    function to double the number
    """
    return 2 * num

doubleNum(10)
```

Out[39]:

```
20
```

In [42]:

```
print(doubleNum.__doc__) #Docstring is available to us as the attribute __doc__
```

```
    function to double the number
```

Python Indentation

1. Most of the programming languages like C, C++, Java use braces { } to define a block of code. Python uses indentation.
2. A code block (body of a function, loop etc.) starts with indentation and ends with the first unindented line. The amount of indentation is up to you, but it must be consistent throughout that block.
3. Generally four whitespaces are used for indentation and is preferred over tabs.

In [44]:

```
for i in range(10):
    print(i)
```

```
0
1
2
3
4
5
6
7
8
9
```

Indentation can be ignored in line continuation. But it's a good idea to always indent. It makes the code more readable.

In [47]:

```
if True:
    print("Machine Learning")
    c = "AAIC"
```

```
Machine Learning
```

In [48]:

```
if True: print("Machine Learning"); c = "AAIC"
```

```
Machine Learning
```

Python Statement

Instructions that a Python interpreter can execute are called statements.

Examples:

In [49]:

```
a = 1  #single statement
```

Multi-Line Statement

In Python, end of a statement is marked by a newline character. But we can make a statement extend over multiple lines with the line continuation character ().

In [50]:

```
a = 1 + 2 + 3 + \
    4 + 5 + 6 + \
    7 + 8
```

In [52]:

```
print(a)
```

```
36
```

In [54]:

```
#another way is
a = (1 + 2 + 3 +
    4 + 5 + 6 +
    7 + 8)
print(a)
```

```
36
```

In [55]:

```
a = 10; b = 20; c = 30   #put multiple statements in a single line using ;
```