Python

Variables

- **Name** which is **used to refer memory location**
- Also **known as identifier** and **used to hold value**
- **No need to specify** the **type** of variable
- Python is a type infer (automatic detection of the data type)language
- Variable **names can be a group of both letters and digits**, but they have to **begin with a letter or an underscore**
- Recommended to **use lowercase letters for variable name**
- E.G. **Marks** and **marks** both are **two different variables**

Identifier Naming

- **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)**.
- **Must not contain** any white-space, or **special character** (!, @, #, %, ^, &, *).
- **Must not be similar to any keyword** defined in the language.
- **Case sensitive**
- Examples of valid identifiers : student12, _max, min_5, etc.
- Examples of invalid identifiers: 5a, x%4, a 9, etc.

Assigning Value to Variable

- **equal (=) operator is used to assign value to a variable**

```
name = “ahir”

marks = 56.76
```

Multiple Assignment

- **Assigning single value to multiple variables**

```
x=y=z=50
```

- **Assigning multiple values to multiple variables**

```
a,b,c=5,10,15
```

Multiple Assignment

- **Assigning single value to multiple variables**

```
x=y=z=50
```

Strings

- Strings can be **created by enclosing the character** or the **sequence of characters in the quotes**.
- Python **allows** to use **single quotes**, **double quotes**, or **triple quotes** to create the string.
- Example
  - text = "Hello there!"
- If we check the type of the variable text using a python script
  - print(type(text)) => then it will print string (str).
- **Strings are treated as the sequence of strings** which means that **python doesn't support the character data type** instead a **single character written as 'r' is treated as the string of length 1**.

Strings indexing and splitting

- **Indexing** of the python strings **starts from 0**.
- Example, The string "HELLO" is indexed as given in the figure.
- the **slice operator [ ]** is used to access the individual characters of the string.

str = "HELLO"

str[0] = 'H'

str[1] = 'E'

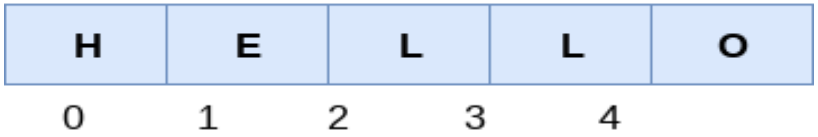
str[2] = 'L'

str[3] = 'L'

str[4] = 'O'

- We can use the **: (colon) operator** in python to **access the substring**.

str = "HELLO"



str[0] = 'H'

str[1] = 'E'

str[2] = 'L'

str[3] = 'L'

str[4] = 'O'

str[:] = 'HELLO'

str[0:] = 'HELLO'

str[:5] = 'HELLO'

str[:3] = 'HEL'

str[0:2] = 'HE'

str[1:4] = 'ELL'

Reassigning strings

- Updating the content of the strings is as easy as assigning it to a new string.
- The **string object doesn't support item assignment**
- i.e., A **string can only be replaced with a new string** since **its content can not be partially replaced**.
- Strings are **immutable** in python.

String Operators

Operator	Description
+	It is known as concatenation operator used to join the strings given either side of the operator.
	str = "Hello" str1 = " world" print(str+str1)# prints Hello world
*	It is known as repetition operator. It concatenates the multiple copies of the same string.
	str = "Hello" str1 = " world" print(str*3) # prints HelloHelloHello
[]	It is known as slice operator. It is used to access the sub-strings of a particular string.
	str = "Hello" print(str[4]) # prints o

Operator	Description
[:]	It is known as range slice operator. It is used to access the characters from the specified range.
	str = "Hello" print(str[2:4]); # prints ll
in	It is known as membership operator. It returns if a particular sub-string is present in the specified string.
	str = "Hello" print('w' in str) # prints false as w is not present in str
not in	It is also a membership operator and does the exact reverse of in. It returns true if a particular substring is not present in the specified string.
	str1 = " world" print('wo' not in str1) # prints false as wo is present in str1.

Operator	Description
r/R	It is used to specify the raw string. Raw strings are used in the cases where we need to print the actual meaning of escape characters such as "C://python". To define any string as a raw string, the character r or R is followed by the string.
	print(r'C://python37') # prints C://python37 as it is written
%	It is used to perform string formatting. It makes use of the format specifiers used in C programming like %d or %f to map their values in python. We will discuss how formatting is done in python.
	str = "Hello" print("The string str : %s"%(str)) # prints The string str : Hello

Python Formatting operator

- Python allows us to use the format specifiers used in C's printf statement.
- Python provides an additional operator % which is used as an interface between the format specifiers and their values.
- It binds the format specifiers to the values.

```
Integer = 10
Float = 1.290
String = "Ayush"
print("Hi I am Integer ... My value is %d\nHi I am float ... My value is %f\nHi I am string ... My value is %s"%(Integer,Float,String));
```

Output:

```
Hi I am Integer ... My value is 10
Hi I am float ... My value is 1.290000
Hi I am string ... My value is Ayush
```

Tuple

- Python Tuple is used to store the **sequence of immutable python objects**.
- Tuple is similar to lists since the value of the items stored in the list can be changed whereas the tuple is immutable and the **value of the items stored in the tuple can not be changed**.
- A tuple can be **written as the collection of comma-separated values** enclosed with the **small brackets**.

```
T1 = (101, "Ayush", 22)
T2 = ("Apple", "Banana", "Orange")
```

Example

```
tuple1 = (10, 20, 30, 40, 50, 60)
print(tuple1)
count = 0
for i in tuple1:
    print("tuple1[%d] = %d"%(count, i));
```

Output:

```
(10, 20, 30, 40, 50, 60)
tuple1[0] = 10
tuple1[0] = 20
tuple1[0] = 30
tuple1[0] = 40
tuple1[0] = 50
tuple1[0] = 60
```

Example

```
tuple1 = tuple(input("Enter the tuple elements ..."))
print(tuple1)
count = 0
for i in tuple1:
    print("tuple1[%d] = %s"%(count, i));
```

Output:

```
Enter the tuple elements ...12345
('1', '2', '3', '4', '5')
tuple1[0] = 1
tuple1[0] = 2
tuple1[0] = 3
tuple1[0] = 4
tuple1[0] = 5
```

If we try to **reassign the items of a tuple**, we would **get an error as the tuple object doesn't support the item assignment**.

An empty tuple can be written as follows.

```
T3 = ()
```

The **tuple having a single value must include a comma** as given below.

```
T4 = (90,)
```

A tuple is indexed in the same way as the lists. The items in the tuple can be accessed by using their specific index value.

Tuple indexing and splitting

The indexing and slicing in tuple are similar to lists. The indexing in the tuple starts from 0 and goes to length(tuple) - 1.

The items in the tuple can be accessed by using the slice operator. Python also allows us to use the colon operator to access multiple items in the tuple.

Tuple = (0, 1, 2, 3, 4, 5)

0	1	2	3	4	5

Tuple[0] = 0

Tuple[1] = 1

Tuple[2] = 2

Tuple[3] = 3

Tuple[4] = 4

Tuple[5] = 5

Tuple[0:] = (0, 1, 2, 3, 4, 5)

Tuple[:] = (0, 1, 2, 3, 4, 5)

Tuple[2:4] = (2, 3)

Tuple[1:3] = (1, 2)

Tuple[:4] = (0, 1, 2, 3)

- tuple elements can be accessed in both the directions.
- The right most element (last) of the tuple can be accessed by using the index -1.
- The elements from left to right are traversed using the negative indexing.

Example.

```
tuple1 = (1, 2, 3, 4, 5)
print(tuple1[-1])
print(tuple1[-4])
```

Output:

```
5
2
```

Tuple operations

The operators like **concatenation (+), repetition (*), Membership (in)** works in the same way as they work with the list.

Let's say Tuple t = (1, 2, 3, 4, 5) and Tuple t1 = (6, 7, 8, 9) are declared.

Operator	Description	Example
Repetition	The repetition operator enables the tuple elements to be repeated multiple times.	T1*2 = (1, 2, 3, 4, 5, 1, 2, 3, 4, 5)
Concatenation	It concatenates the tuple mentioned on either side of the operator.	T1+T2 = (1, 2, 3, 4, 5, 6, 7, 8, 9)
Membership	It returns true if a particular item exists in the tuple otherwise false.	print (2 in T1) prints True.
Iteration	The for loop is used to iterate over the tuple elements.	for i in T1: print(i) **Output** 1 2 3 4 5
Length	It is used to get the length of the tuple.	len(T1) = 5

Tuple inbuilt functions

SN	Function	Description
1	cmp(tuple1, tuple2)	It compares two tuples and returns true if tuple1 is greater than tuple2 otherwise false.
2	len(tuple)	It calculates the length of the tuple.
3	max(tuple)	It returns the maximum element of the tuple.
4	min(tuple)	It returns the minimum element of the tuple.
5	tuple(seq)	It converts the specified sequence to the tuple.

List

- To store the **sequence of various type of data**.
- python contains six data types that are capable to store the sequences but the most common and reliable type is list.
- A list can be defined as a **collection of values or items of different types**. The items in the list are separated with the comma (,) and enclosed with the **square brackets [].**

A list can be defined as follows.

L1 = ["John", 102, "USA"]

L2 = [1, 2, 3, 4, 5, 6]

L3 = [1, "Ryan"]

If we try to print the type of L1, L2, and L3 then it will come out to be a list.

List indexing and splitting

- The elements of the list can be accessed by using the **slice operator []**.
- The index starts from 0 and goes to length - 1.
- The first element of the list is stored at the 0th index, the second element of the list is stored at the 1st index, and so on.

List = [0, 1, 2, 3, 4, 5]

0	1	2	3	4	5

List[0] = 0

List[1] = 1

List[2] = 2

List[3] = 3

List[4] = 4

List[5] = 5

List[0:] = [0,1,2,3,4,5]

List[:] = [0,1,2,3,4,5]

List[2:4] = [2, 3]

List[1:3] = [1, 2]

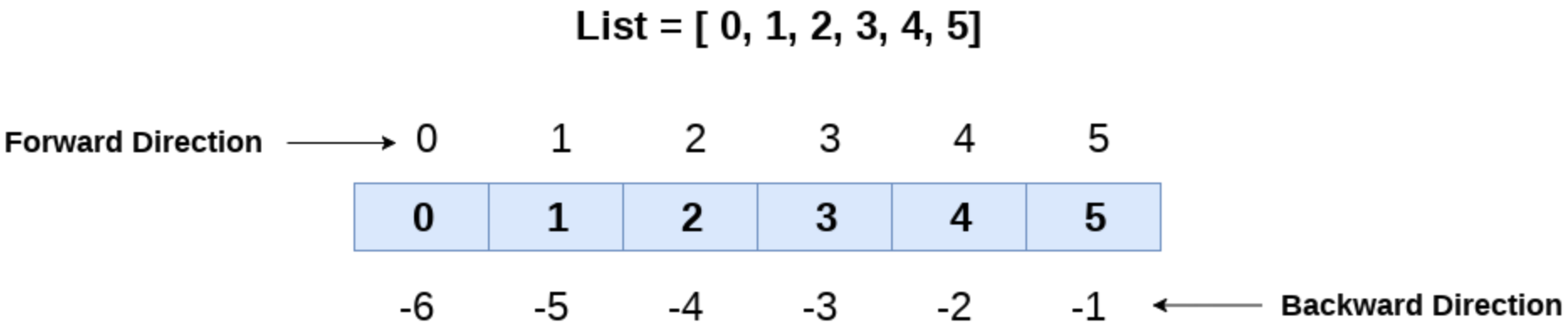
List[:4] = [0, 1, 2, 3]

Python provides us the flexibility to use the negative indexing also.

The negative indices are counted from the right.

The last element (right most) of the list has the index -1, its adjacent left element is present at the index -2 and so on until the left most element is encountered.

Python Lists



Updating List values

- Lists are immutable and their values can be updated by using the slice and assignment operator.
- Python also provide us the append() method which can be used to add values to the string.

```
List = [1, 2, 3, 4, 5, 6]
print(List)
List[2] = 10;
print(List)
List[1:3] = [89, 78]
print(List)
```

Output:

```
[1, 2, 3, 4, 5, 6]
[1, 2, 10, 4, 5, 6]
[1, 89, 78, 4, 5, 6]
```

Deleting List values

List elements can be deleted by using the **del** keyword.
Python also provides us the remove() method if we do not know which element is to be deleted from the list.

```
List = [0,1,2,3,4]
print(List)
del List[0]
print(List)
del List[3]
print(List)
```

Output:

```
[0, 1, 2, 3, 4]
[1, 2, 3, 4]
[1, 2, 3]
```

Python List Operations

- The concatenation (+) and repetition (*) operator work in the same way as they were working with the strings.
- Lets see how the list responds to various operators.
- Consider a List l1 = [1, 2, 3, 4], and l2 = [5, 6, 7, 8]

Operator	Description	Example
Repetition	The repetition operator enables the list elements to be repeated multiple times.	L1*2 = [1, 2, 3, 4, 1, 2, 3, 4]
Concatenation	It concatenates the list mentioned on either side of the operator.	l1+l2 = [1, 2, 3, 4, 5, 6, 7, 8]
Membership	It returns true if a particular item exists in a particular list otherwise false.	print(2 in l1) prints True.
Iteration	The for loop is used to iterate over the list elements.	for i in l1: print(i) **Output** 1 2 3 4
Length	It is used to get the length of the list	len(l1) = 4

Iterating a List

A list can be iterated by using a **for - in** loop. A simple list containing four strings can be iterated as follows.

```
List = ["John", "David", "James", "Jonathan"]
for i in List:
    print(i)
```

Output:

```
John
David
James
Jonathan
```

Adding elements to the list

- Python provides **append()** function by using which we can **add an element** to the list.
- append() method can only add the value to the end of the list.

Example

```
l =[ ]
n = int(input("Enter the number of elements in the list"))
for i in range(0,n):
    l.append(input("Enter the item: "))
print("printing the list items....")
for i in l:
    print(i)
```

Removing elements from the list

```
List = [0,1,2,3,4]
print("printing original list: ");
for i in List:
    print(i,end=" ")
List.remove(0)
print("\nprinting the list after the removal of first element...")
for i in List:
    print(i,end=" ")
```

Output:

printing original list:

0 1 2 3 4

printing the list after the removal of first element...

1 2 3 4

Python List Built-in functions

SN	Function	Description
1	cmp(list1, list2)	It compares the elements of both the lists.
2	len(list)	It is used to calculate the length of the list.
3	max(list)	It returns the maximum element of the list.
4	min(list)	It returns the minimum element of the list.
5	list(seq)	It converts any sequence to the list.

SN	Function	Description
1	list.append(obj)	The element represented by the object obj is added to the list.
2	list.clear()	It removes all the elements from the list.
3	List.copy()	It returns a shallow copy of the list.
4	list.count(obj)	It returns the number of occurrences of the specified object in the list.
5	list.extend(seq)	The sequence represented by the object seq is extended to the list.
6	list.index(obj)	It returns the lowest index in the list that object appears.
7	list.insert(index, obj)	The object is inserted into the list at the specified index.
8	list.pop(obj=list[-1])	It removes and returns the last object of the list.
9	list.remove(obj)	It removes the specified object from the list.
10	list.reverse()	It reverses the list.
11	list.sort([func])	It sorts the list by using the specified compare function if given.

Nesting List and tuple

We can store list inside tuple or tuple inside the list up to any number of level.

Example

```
Employees = [(101, "Ayush", 22), (102, "john", 29), (103, "james", 45), (104, "Ben", 34)]
print("----Printing list----");
for i in Employees:
    print(i)
Employees[0] = (110, "David",22)
print()
print("----Printing list after modification----");
for i in Employees:
    print(i)
```

List VS Tuple

SN	List	Tuple
1	The literal syntax of list is shown by the [].	The literal syntax of the tuple is shown by the ().
2	The List is mutable.	The tuple is immutable.
3	The List has the variable length.	The tuple has the fixed length.
4	The list provides more functionality than tuple.	The tuple provides less functionality than the list.
5	The list Is used in the scenario in which we need to store the simple collections with no constraints where the value of the items can be changed.	The tuple is used in the cases where we need to store the read-only collections i.e., the value of the items can not be changed. It can be used as the key inside the dictionary.