PYTHON

Session - 1

Print("Hello World")
↓ function
↓ string ("", '', """ """)
→ Argument

'cloud camp is 'excellent' → Syntax error

"cloud camp is 'excellent'" → Correct syntax

""" """ → triple quotes → Used for multiline string

```
print("""cloud camp
          is
      'excellent'""")
```

o/p = cloud camp
is
'excellent'

"cloud camp is "excellent"" → syntax error

So, we use Escape Sequences.

Escape Sequences -

"cloud camp is \"excellent\" "
escape sequences

- PYTHON is Case Sensitive

Session - 2

Variables
Datatypes

Numbers -

Integers - -1, 0, 1, etc
Float - 7.55, 3.14, 2.86, etc.
Complex numbers - 1+2i, 3+4i, etc.
String - 'a', "abc", """a
b
c""", '''

Boolean datatypes
true or false
(1) (0)

Datatypes -

int string
float bool
complex (rarely usage)

Python specific datatypes -

list dict
tuple set

Print (5**2) → 25
Print (5**5) → 3125
Print (1+1) → 2
Print (10-5) → 5
Print (10*5) → 50
Print (10/5) → 2.0 (default : float)
Print (10//5) → 2 (integer division)
Print (10%5) → 0 (Modulo operator - %)
Print ('a'+'b') → ab (addition of 2 strings)
↑
string Concatenation
Print ("cloud" + "camp") → cloudcamp
Print ("cloud" + " " "camp") → cloud camp
↓
Character or literal
Print ("cloud clamp") → cloud camp
print ("cloud"_"camp") → Syntax error.
print ("cloud"/"camp") → syntax error.
print ("cloud" * "camp") → syntax error.

** - exponent | power
/ - quotient
% - remainder

Operator Precedence [tells which operator to perform first]

BODMAS → Brackets | Of | Division | Multiplication | Addition | Subtraction.

Variables - Place holders - identifier

Ex - Area of a circle = πR^2

pi = 3.14
radius = 5
area = pi * radius * * 2
↓
expression

$R^2 \rightarrow R*R$
$10^{10} \rightarrow R**10$.

Program

Pi = 3.14
radius = 5
area = pi * radius **5
or
pi * (radius **5)
print (area)
o/p ⇒ 78.5

Comments in Python

- Comments are for user information / programmer info
- # comment
- ignores at the time of execution.

Types of comments

- Normal comments
- In line comments.

session - 3

- type, id functions
- other datatypes : mutable and immutable

type() → what is a datatype of the variable

Ex - 1) print(type(12.5)) ⇒ <class 'float'>
2) print(type("cloudcamp")) ⇒ <class 'str'>
3) print(type("true")) ⇒ <class 'str'>
4) print(type(true)) ⇒ class 'bool'>

id() → It represents the address / memory location of the variable.

Ex - 1) a = 12.5
print(id(a))
O/p = 1234567890 (Ram location).

Variable Naming Conventions -

Rules -

- start with a character.
- '_' underscore can be a starting letter.
- all lower case (or) all upper case (used for constant) (or) Camel case
  (or) Snake case
  ↓
  first_name
  ↓
  seperate with underscore
  ↓
  FirstName

- digits can be used or added

eg - Person 1, car 1 ✓

but not 1 person or 1 car ✗

- $ cannot be used (dollar).

Comparision Operators -

>, <, >=, <=, !=, ==

Result = Boolean (True or false)

(1) (0)

Ex -

```
a = 2, b = 3
print(a<b)
print(a<=b)
o/p = True
      True
```

```
a = 2
b = 3
print(a>b)
print(a<=b)
o/p = false
      true
```

Logical Operators -

and, or, not

Ex - 1) a = 2 (OR)
b = 3
print((a>b) or (b>a))
o/p = TRUE

2) a = 2 (AND)
b = 3
print((a>b) and (b<a))
o/p = FALSE

OR

a	b	o/p
0	0	0
0	1	1
1	0	1
1	1	1

AND

a	b	o/p
0	0	0
0	1	0
1	0	0
1	1	1

Assignment Operators -

+=, -=, *=, /=

Ex - 1) a = 2
print(a+=2) #a=a+2
o/p = 4

2) a = 2
print(a-=2) #a-2
o/p = 0

3) a = 2
print(a*=2) #a*2
o/p = 4

4) a = 2
print(a/=2) #a/2
o/p = 1.0

Mutability and Immutability -

Mutable - Once defined, can be changed later.

Immutable - Once defined, cannot be changed later.

Input - How to take an input from user

Ex - name = input()
print(name)

o/p - cursor blinks for user to give an input.

(or)

name = input("Please enter your name:")
print("Hello", +name)

o/p - Please enter your name : Pavan
Hello, Pavan.

(or)

name = input("please enter your name:")
print("Hello", +name)

o/p - Please enter your name : 123
Hello, 123.

input → by default → string → stores and returns a string.

To know the datatype of name -

```
name = input("please enter your name:")
print(type(name))
print("Hello", name)
```

o/p - Please enter your name : 123
<class 'str'>
Hello, 123

Ex -
```
a = input("enter a number = ")
b = input("enter a number = ")
print(a+b)
```

o/p - enter a number = 2
enter a number = 3
23
enter a number = cloud
enter a number = camp
cloudcamp.

Datatype : list –

→ Using list, we can store multiple values in a variable.
→ defined using square brackets [ ]

```
a = [1,2,3,4,5]
a = [1,2,3, "cloud", "camp"]
a = [1,2,3, "cloud", True].
```

Indexing : Accessing elements inside a variable
→ element position inside a variable / indices / index.
→ zero indexing [starts with zero].

```
 0  1  2    3      4   ⟶ Index.
[1, 2, 3, "cloud", True] ⟶ List
```

how to access ⇒ a[ ]
↑
Position / index

Ex – print(a[0]) ⟶ 1
print(a[3]) ⟶ cloud.

List example [+Ve Indexing o/p]

ex – a = [1, 2, 3, "cloud", True]

```
print(a[0])
print(a[1])
print(a[2])
print(a[3])
print(a[4])
```

o/p ⇒
1
2
3
cloud
True

```
print(a[5])
```

o/p ⇒ indexerror : list index out of range

Indexing → +ve index & -ve index

List example (-ve indexing o/p)

Ex - a = [1, 2, 3, "cloud", True]
(-5 -4 -3 -2 -1)

```
print(a[-1])
print(a[-2])
print(a[-3])
print(a[-4])
print(a[-5])
```

o/p ⇒ True
cloud
3
2
1

How do you find the length of a list?

↓

No. of elements in a list.

```
a = [1, 2, 3, "cloud", True]
print(len(a)) ⇒ 5.
print("length of a list:", len(a))
```

o/p ⇒ length of a list : 5.

Variable Naming Convention (as per company standard)

1) Camel Case ↓ Ex - varName

2) Snake Case Ex - var_name (no caps).

3) Pascal case Ex - VarName

⇒ print(a, len(a))

o/p ⇒ [1, 2, 3, "cloud", True] 5.

→ List is a heterogenous data type (holds many or any data).

a = [10, 'a', True]

- Slicing operation in list
- Appending elements to list
- Extending a list
- Insert an element
- Update a list
- delete a list

List slicing - to display many elements

a = [1,2,3,4,5,6] | a[startindex : endindex]
a[0] = 1 ↑ not included.

a[0:3] = 1,2,3
Print (a[0:3]) ⇒ [1,2,3]
ex = a = [1,2,3,4,5] → by default starting index=0
Print(a[0:3]) | a[:3] | a[0:] → by default ending index.
o/p ⇒ [1,2,3]

a = [1, 2, 3, 4, 5]
print (len [a]) ⇒ 5 (No. of elements)
print [a[length(a)] X list is out of range.
print [a[len(a)-1] ⇒ 1,2,3,4,5.

⇒ a = [1, 2, 3, 4, 5]
print (a[::2] → 0:len(a):2
↑ step size

o/p ⇒ 1, 3, 5.

- Fetch all the elements inside a list that are at even positions.
  print(a[::2]) ⇒ 1, 3, 5.
- Fetch all the elements inside a list that are at odd positions.
  print(a[1::2]) ⇒ 2, 4.

a = [10, 20, 30, 40, 50]

print(a[-1:-3]) | a[-3:-1] | print(a[-1::-1])

o/p → [ ] | o/p → [30, 40] | o/p → [50, 40]

Reversing a list -

a = [10, 20, 30, 40, 50]

Print(a[::-1])

o/p → 50, 40, 30, 20, 10

Appending and Extending (Adds elements at the end of the list only).

- Adding / Appending values to a list

Ex - a = [10, 20, 30, 40, 50] ← 60

a.append(60) → this affects the original list i.e changes are made to the original list.

print(a)

o/p → [10, 20, 30, 40, 50, 60]

a.extend(70) ← affects the original

print(a)

o/p → [10, 20, 30, 40, 50, 60, 70] ✗

a = [10, 20, 30, 40, 50]

a.append(25, 55) ✗

a.append([25, 55]) ⇒ [10, 20, 30, 40, 50, [25, 55]]
0 1 2 3 4 5

a.extend([25, 55]) ⇒ [10, 20, 30, 40, 50, 25, 55]
0 1 2 3 4 5 6

- Both append and extend adds element to the end of the list

How to insert an element as per user choice?

- insert(,)
  ↑ Position ↑ value

Ex - a = [10, 20, 30, 40, 50] len = 5

a.insert(1, 100)

print(a)

o/p ⇒ [10, 100, 20, 30, 40, 50] len = 6

How to update an element in a list?

Ex ⇒ a = [10, 20, 30, 40, 50]

a[1] = 200 # updating the element 20 to 200

print(a)

o/p ⇒ 10, 200, 30, 40, 50

a [10, 20, 30, 40, 50, [25, 55]]

a[5] = [25, 55]

a[5][0] = 25

a[5] → subset a mainlist

[0] → accessing an element from sublist

Fetch the element 8 from the following list

a = [10, 20, 30, 40, 50 [25 [50, 60, 15 [10, 8, 6]]]]

① ② ③ ④

1) 4D list

2) Length = 6

a[5] = [25, [50, 60, 15 [10, 8, 6]]]

3D list

a[5][1] = [50, 60, 15 [10, 8, 6]]

2D list

a[5][1][3] = [10, 8, 6]

1D list

print(a[5][1][3][1]) ⟹ 8

Deleting elements from a list

a = [10, 20, 30, 40, 50]

1) Pop() ↓ removes the element based on the index and returns that element (only one element)

2) remove() ↓ removes the element based on the value and it cannot return the element It tells None.

3) clear ↓ removes all the elements in a list and returns an empty list and can't return values It returns None

4) del → removes the reference of the list

pop()

```
a = [10,20,30,40,50]
    0  1  2  3  4
a.pop(1)
print(a)
```
o/p ⇒ [10,30,40,50]

```
a = [10,20,30,40,50]
b = a.pop(1)
print(a)
print(b)
```
o/p ⇒ [10,30,40,50]
20

```
a = [10,20,30]
b = a.pop()
print(a)
print(b)
```
o/p ⇒ [10,20]
30
By default it remove last element

remove()

```
a = [10,20,30,40,50]
    0  1  2  3  4
a.remove(30)
print(a)
```
o/p ⇒ [10,20,40,50]

```
a = [10,20,30,40,50]
b = a.remove(20)
print(a)
print(b)
```
o/p ⇒ [10,30,40,50]
None

```
a = [1,2,3,4,5,6]
b = a.remove(8)
print(a)
print(b)
```
[1,2,3,4,5,6]
o/p ⇒ x not in the list

Clear

```
a = [25,42,89,64,12]
b = a.clear()
print(a)
print(b)
```
o/p ⇒ [ ]
None

How to sort a list ?

1) Sort (inplace operation) 2) Sorted

```
a = [5,3,1,2,4]
a.sort()
print(a)
```
o/p ⇒ [1,2,3,4,5]

Ascending

```
a = [5,3,1,2,4]
b = sorted(a)
print(a)
print(b)
```
o/p ⇒ [5,3,1,2,4]
[1,2,3,4,5]

strings -

```
a = ['w','a','t','u','z']
a.sort()
print(a)
```
o/p ⇒ ['a','t','u','w','z']

```
a = ['w','a','t','u','z',2,1,10]
a.sort()
print(a)
```
o/p ⇒ not supported between instances of 'int' and 'string'

```
a = ['a','A','b','c']
a.sort() // based on ASCII values.
print(a)
```
o/p ⇒ ['A','c','a','b']

Descending

```
a = [5,3,1,4,2]
b = sorted(a, reverse = True)
print(a)
print(b)
```

o/p ⇒ [5,3,1,4,2]
[5,4,3,2,1]

```
a = [5,3,2,4,1]
b = sorted(a, reverse = False)
a.sort() # ascending
a[::-1] # reversing
print(b)
```

} descending

o/p ⇒ [5,4,3,2,1]

Index of an element in a list

```
a = [10,20,30,40,50]
a_id = a.index(30), value
print(a_id)
```

o/p ⇒ 2

```
a = [10,20,30,20,40,20,50]
a_id = a.index(20)
print(a_id)
```

Default : first occurance element index

o/p ⇒ 1

```
a = [10,20,30]
a_id = a.index(250)
print(a_id)
```

o/p ⇒ 250 is not in list

Count the occurence of an element in a list

```
a = [10,20,30,20,10,20,50]
b = a.count(20)
print(b)
```

o/p ⇒ 3

```
a = [10,20,50]
b = a.count(80)
print(b)
```

o/p ⇒ 0

How to add 2 lists ?

```
l1 = [10,20,30,40,50]
l2 = [1,2,3,4,5]
l = list() # empty list
l.append(l1)
l.append(l2)
l.extend((l1,l2))
print(l, len(l))
```

o/p ⇒ [[10,20,30,40,50],[1,2,3,4,5]] 2

```
l1 = [10,20,30,40,50]
l2 = [1,2,3,4,5]
l = list()
l.extend(l1)
l.extend(l2)
print(l, len(l))
```

o/p ⇒ [10,20,30,40,50,1,2,3,4,5] 10

```
a = [10,20,30,40,50]
b = [1,2,3,4,5]
c = a+b
print(c)
```

o/p ⇒ [10,20,30,40,50,1,2,3,4,5]

```
l = [1,2,3,4,5]
l = l+l (or) l*2
print(l)
```

o/p ⇒ [1,2,3,4,5,1,2,3,4,5]