MODULE-1

PPT-4

ASCII characters

- To find the ASCII value of a character, we can use the **ord() function**, which is a built-in function in Python that accepts a char (string of length 1) as argument and returns the unicode code point for that character.
- Example-

```
a=‘d’
ord(a)
ord(‘A’)
```

- We can also find the character from a given ASCII value using **chr() function**. This function accepts the ASCII value and returns the character for the given ASCII value.

```
x=97
print(chr(x))
print(chr(97))
```

Python:Boolean

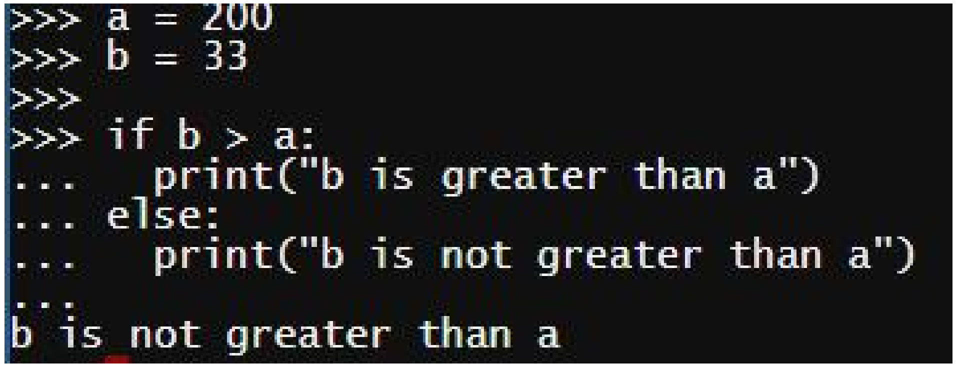
- In programming you often need to know if an expression is True or False.
- You can evaluate any expression in Python, and get one of two answers, True or False.
- When you compare two values, the expression is evaluated and Python returns the Boolean answer. Example

```
print(10 > 9) #will generate TRUE
print(10 == 9) #False
print(10 < 9)#False
```

- When you run a condition in an if statement, Python returns True or False:

- Example-

```
a = 200
b = 33
if b > a:
  print("b is greater than a")
else:
  print("b is not greater than a")
```



```
>>> a = 200
>>> b = 33
>>>
>>> if b > a:
...     print("b is greater than a")
... else:
...     print("b is not greater than a")
...
b is not greater than a
```

- The bool() function allows you to evaluate any value, and give you True or False in return

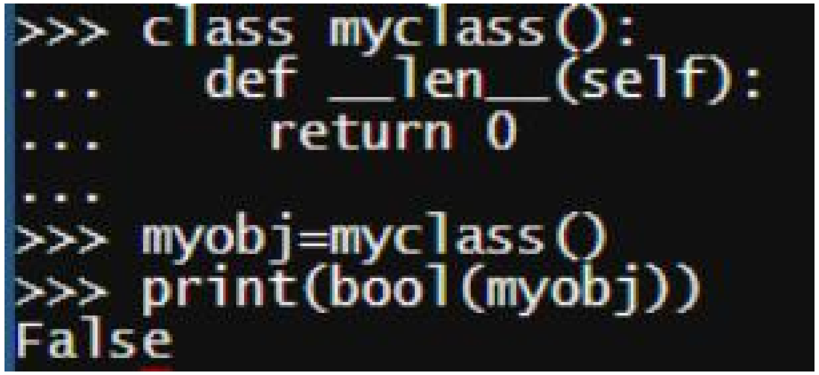
```
print(bool("Hello"))
print(bool(15))
print(bool())
print(bool(''))
print(bool('a'))
```

- Almost any value is evaluated to True if it has some sort of content.
- Any string is True, except empty strings.
- Any number is True, except 0.
- Any list, tuple, set, and dictionary are True, except empty ones.
- In fact, there are not many values that evaluates to False, except empty values, such as (), [], {}, "", the number 0, and the value None. And of course the value False evaluates to False.

```
print(bool(False))
print(bool(None))
print(bool(0))
print(bool(""))
print(bool(()))
print(bool([]))
print(bool({}))
```

- One more value, or object in this case, evaluates to False, and that is if you have an object that is made from a class with a __len__ function that returns 0 or False:

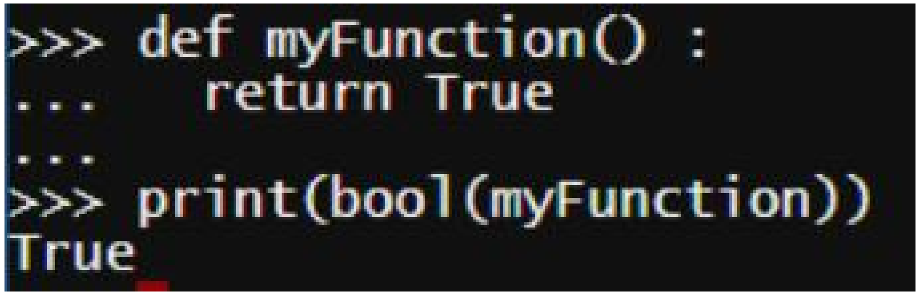
```
>>>class myclass():
…       def __len__(self):
…         return 0
…
>>>myobj = myclass()
>>>print(bool(myobj))
```



```
>>> class myclass():
...     def __len__(self):
...       return 0
...
>>> myobj=myclass()
>>> print(bool(myobj))
False
```

- You can create functions that returns a Boolean Value:

```
>>> def myFunction() :
...      return True
...
>>> print(bool(myFunction))
```



```
>>> def myFunction() :
...     return True
...
>>> print(bool(myFunction))
True
```

- Python also has many built-in functions that returns a boolean value, like the **isinstance()** function, which can be used to determine if an object is of a certain data type:

```
>>> x = 200
>>> print(isinstance(x, int))
>>> y = "priyanka“
>>> print(isinstance(y, str))
>>> print(isinstance(y, int))
```

Python Operators

- Operators are used to perform operations on variables and values.
- Python divides the operators in the following groups:
  1. Arithmetic operators
  2. Assignment operators
  3. Comparison operators
  4. Logical operators
  5. Identity operators
  6. Membership operators
  7. Bitwise operators

Python Arithmetic Operators

Arithmetic operators are used with numeric values to perform common mathematical operations:

Operator	Name	Example
+	Addition	x + y
-	Subtraction	x - y
*	Multiplication	x * y
/	Division	x / y
%	Modulus	x % y
**	Exponentiation	x ** y
//	Floor division	x // y

Python Assignment Operators

Operator	Example	Same As
=	x = 5	x = 5
+=	x += 3	x = x + 3
-=	x -= 3	x = x - 3
*=	x *= 3	x = x * 3
/=	x /= 3	x = x / 3
%=	x %= 3	x = x % 3
//=	x //= 3	x = x // 3
**=	x **= 3	x = x ** 3
&=	x &= 3	x = x & 3
|=	x |= 3	x = x | 3
^=	x ^= 3	x = x ^ 3
>>=	x >>= 3	x = x >> 3
<<=	x <<= 3	x = x << 3

Assignment operators are used to assign values to variables.

Python Comparison Operators

Comparison operators are used to compare two values.

Operator	Name	Example
==	Equal	x == y
!=	Not equal	x != y
>	Greater than	x > y
<	Less than	x < y
>=	Greater than or equal to	x >= y
<=	Less than or equal to	x <= y

Python Logical Operators

Logical operators are used to combine conditional statements.

Operator	Description	Example
and	Returns True if both statements are true	x < 5 and x < 10
or	Returns True if one of the statements is true	x < 5 or x < 4
not	Reverse the result, returns False if the result is true	not(x < 5 and x < 10)

Python Identity Operators

- Identity operators are used to compare the objects, not if they are equal, but if they are actually the same object, with the same memory location. Used for list datatype
-

Operator	Description	Example
is	Returns True if both variables are the same object	x is y
is not	Returns True if both variables are not the same object	x is not y

Example

```
x = ["apple", "banana"]
y = ["apple", "banana"]
z = x
print(x is z)
# returns True because z is the same object as x
print(x is y)
# returns False because x is not the same object as y, even if
they have the same content
print(x == y)
# to demonstrate the difference betweeen "is" and "==": this
comparison returns True because x is equal to y
```

Python Membership Operators

Membership operators are used to test if a sequence is presented in an object:

Operator	Description	Example
in	Returns True if a sequence with the specified value is present in the object	x in y
not in	Returns True if a sequence with the specified value is not present in the object	x not in y

Example:

```
x = ["apple", "banana"]
print("banana" in x)
# returns True because a sequence with the value "banana" is in the list
```

Python Bitwise Operators

- Bitwise operators are used to compare (binary) numbers.

Operator	Name	Description	Syntax
&	AND	Sets each bit to 1 if both bits are 1	x & y
|	OR	Sets each bit to 1 if one of two bits is 1	x | y
^	XOR	Sets each bit to 1 if only one of two bits is 1	~x
~	NOT	Inverts all the bits () Returns one’s complement of the number.	x ^ y
>>	Bitwise right shift	Shifts the bits of the number to the right and fills 0 on voids left as a result.	x>>
<<	Bitwise left shift	Shifts the bits of the number to the left and fills 0 on voids left as a result.	x<<