Python Objects

→ Python is an object-Oriented prgming lang.
→ Almost everything in python is an object, with its properties and methods.

A class is like an object constructor, or a blueprint for creating objects.

Create a class :—

↳ To create a class, use the keyword "class"

Eg:- class Myclass:
x = 5

Create Object :-

We can use the class named Myclass to create objs.

```
P1 = Myclass()
print(P1.x)
```

The __init__() Function :-

→ *Above class & objects are not really useful in real-time applications.

→ To understand the meaning of classes we have to understand the built-in __init__() fn.

→ __init__(), which is always executed when the class is being initiated.

→ Use the __init__() fn. to assign values to object properties, when the obj. is being created.

Example:-

```
class person:
    def __init__(self, name, age):
        self.name = name
        self.age = age

P1 = person("Varun", 21)
print(P1.name)
print(P1.age)
```

Note:- The __init__() fn is called automatically everytime the class is being used to create a new object.

Note:- The 'self' parameter is a reference to the current instance of the class, and is used to access variables that belong to the class.

→ It does not have to be named self, we can use (call) whatever we like, but it has to be the first parameter

Eg:-
```
class person:
    def __init__(myobject, name, age):
        myobject.name = name
        myobject.age = age
    def myfun(abc):
        print("Hello my name is" + abc.name)

P1 = person("Teja", 25)
P1.myfun()
```
(self)

O/p:- Hello my name is Teja

```
P1.age = 40
print(P1.age)
```
→ O/p:- 40

```
del P1.age
print(P1.age)
```
O/p:- 'person' object has no attribute 'age'

```
del P1
print(P1)
```
O/p:- P1 is not defined

pass stmt:-

Class definitions can't be empty, but if you for some reason have a class defn with no content, put in the pass stmt, to avoid getting an error

eg:- class person:
    pass

Expressions:-

Values:- values can be integers, strings, boolean & floating pnt number

2 - integer, hello - string, 5.2 - float, T/F - bool

Type:- type(2) → int
type("hello") → str
type(5.2) → float
type(True) → bool

Variables:- A name that refers to a value.

Note that the values of variables can change, that's why they're called variables.

eg:- val = 2, val1 = 5.6, val2 = ('hello') val3 = Fal

type(val) → int

Variable Names And keywords

- Can have letters, numbers and underscore
- Can begin with letter and underscore
- Case sensitive
- does not contain keywords (Reserved words)
- does not contain special symbols.

eg:- in_01 = 10 → print(in_01) → 10
_input02 = 20 → 20

"as" does not keyword

Eg:- assert = True
↓
Syntax Error

input_05@ = 50
↓
Syntax Error

input_06 = 100
↓
Space Error

* Keywords (33) keywords available in python

Statements :-

- Instruction that python interpreter can execute
- Two kinds of stmts print and assignment
- Result of print is stmt with a value.

Eg:- print (100) → 100

print (<expr>, <expr>, ..., <expr>)

print (100+200) → 300

print (6, 7, 8) → 6, 7, 8

```
print(9)
print(10)
```
→ 9
10

print ("show the result in same line", end = " ")
print ("This string will follow in same line")
→ output will print in same line

The format method :-

- Sometimes we may want to construct strings from other info".
- This is where the format() method is useful.

Eg: print ("{ }, example of format method".format ("first"))

Eg:- str = 'I am working on { }'
print (str.format ("python 3.8"))
o/p:- I am working on python 3.8

print("This is my {} tutorial, {} was on {}".
format(1,2,"Anconda python"))

different place holders.

eg:1) print("Name : {name}, Age : {age}".format(name=
"David", age=30))

2) print("Name : {0}, Age : {1}".format("John", 30))

3) print("Name : {}, Age : {}".format(" "))

* Assignment statement does not produce any result.

```
a = 5+7          |  a = 10
a                |  a = a+1
7                |  a ⟶ 11
```

Simultaneous Assignment

1) a, b, c = 100, 200, 300
print(a, b, c) ⟶ 100 200 300

2)
```
u = 10                      z = 10
v = 20          ⟹          w = 20
t = u                       z, w = w, z
u = v                       print(z, w) ⟶ 20, 10
v = t
print(u, v) ⟶ 20 10
```

Evaluating Expression

- An expression is a combination of values, variables and operators.
- The fragment of prgm code that produce or calculate new data values are called expressions.
- The simplest kind of exprn is a literal

- A simple identifier can also be an expression.
- More complex and interesting expressions can be constructed by combining simpler expressions with operators.

Eg:- 2 ⟶ 2 , a = 10 → a → 10
a + 2 → 7

Order of Operation (Rules of precedence)

- parantheses have the highest precedence
- Exponentiation has the next
- multiplication & Division have same pre, which is higher than Addition and subtraction
- Operators with the same precedence are evaluated from left to right.

Eg:- a = 1 , b = 2 , c = 3 , d = 4

e = (a+b) * c // d ⇒ e ⇒ 2

e = (a+b) * (c // d) ⇒ e ⇒ 0

e = a + (b * c) // d ⇒ e ⇒ 2

print (2 ** 3 ** 2) ⇒ 512

2 ** 9 ← 9
512 ←

print ((2 ** 3) ** 2) ⇒ 64

Operations on strings :—

python also provides operators for string

str1 = "python"
str2 = "prog"
str1 + " " + str2 ⇒ python prog
"python" * 3 ⇒ python python python

Triple Quotes :-

- you can specify multi-line strings using triple quotes. (''' or """)

eg:- str1 = """ This is multi-line string.
first line
pytho pr """

Escape sequence :-

1) 'what\'s your name?'

'what's your name?

Line continuation :-

Eg:-2) "This is first.\ ⟹ This is first This is the secnd
This is the secnd"

Strings are immutable

str1 = "python"

str[0] = "c"
↳ str obj does not support item assign

str2 = "c" + str1[1:]
str2 ⟹ cython

String slicing examples

str2 = str1[:5] → pytho

method
find()
replace()
:

Composition :-

One of the most useful feature of prgming languages is their ability to take small building blocks and compose them.

Eg:- c = 2+5
1) print ("Expresn is {}".format(c)) ⟹
Expresn is 7

Operators in Python

i) Arithmetic operators — +, -, *, /, %, //, **
ii) Relational operators — !=, ==, <, <=, >, >=
iii) Unary operators —— unary (+), unary minus (-)
iv) Bit-wise operators —— !, &, ^, ~, <<, >>
v) Logical " —— &&, !!, !
vi) Assignment " —— =, +=, -=, *=, /=, //= ...
vii) Membership " —— in, not in
viii) Identity " —— is, is not

i) Arithmetic operators

a = 5, b = 3

a + b → 8
a - b → 2
a * b → 15
a / b → 1.6
a // b → 1
a % b → 2
a ** b → 5^3 → 125

ii) Relational operators

a = 5, b = 3

a != b → True
a == b → False
a < b → False
a <= b → False
a > b → True
a >= b → False

iii) Unary operators

a = 5,

+a → 5
-a → -5

(iv) Bit-wise operators (performed on Binary numbers)

Bitwise OR (|) → Both are False – False
otherwise – True

Bitwise AND (&) → Both are True – True
otherwise – False

Bitwise XOR (^) → Both are True/False – False
otherwise – True

Complement (~) → True → False, False → True

leftshift (<<) → Shifts the no. of bits towards left

Right shift (>>) → shifts the no. of bits towards right

Eg:- (!) a = 5 → 0000 0101
b = 3 → 0000 0011
} Result 00000111 →

(&) Result → 0000 0001 → 1

(^) Result → 0000 0110 → 6

(~) Result → (~5) → 1111 1010 → 250

(<<) Result → a<<2 → 0000 0101 <<2
00010100 → 20

(>>) Result → a>>2 → 00000001 → 1

(v) Logical Operators :- (whenever we compare t
conditions we go for lo

Logical AND (&&) → Both are True → True
otherwise – False

Logical OR (||) → Both are False → False
otherwise → True

Logical NOT (!) → True → False
False → True

Eg:- (a > b) && (a > c)
T && T → T
F && T → F

(vi) Assignment operator :-

Assignment operator ($=$) → $a = 10$, $a = b$

$+=$ → $a += 1$ → 11

$-=$ → $a -= 1$ → 10

$*=$ → $a *= 2$ → 20

(vii) Membership operators :- (results in bool)

in — variable in sequence
- ① List
- ② Tuple
- ③ Dictionary

↳ if Exists → True - 1
not Exists → False - 0

not in
↳ variable not in sequence
↳ if not exists — True - 1
otherwise → False - 0

(viii) Identity operators :-

is → If two variables reference to same object return True, otherwise False

$a = 5$, $b = 6$ } a is b → False

$a = 'Hyd'$, $b = 'Hyd'$ } a is b → True

is not → If two variables not referencing to the same object - True, otherwise - False

$a = 5$, $b = 6$ } a is not b → True