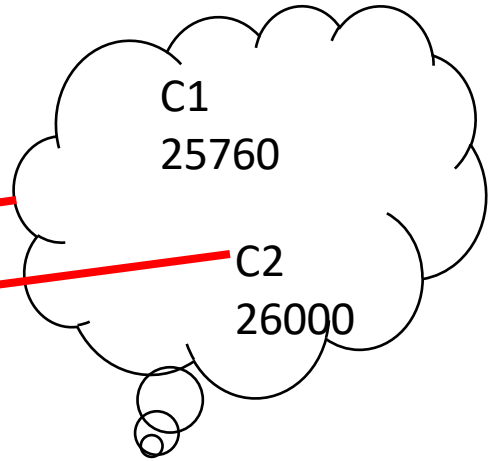
Classes and Object

Memory Allocation

```
class Computer:
    pass

c1=Computer()
c2=Computer()

print(id(c1))
print(id(c2))
```

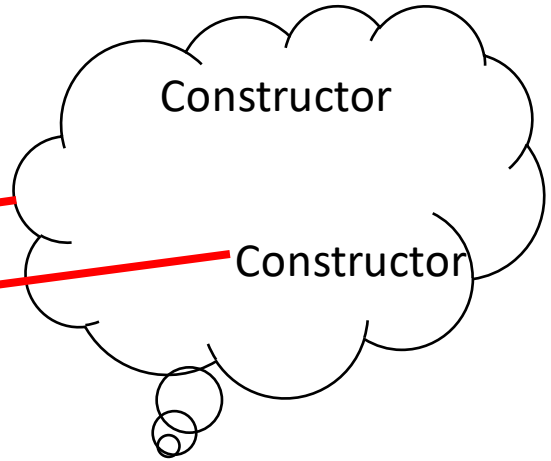



```
140289883425760
140289883426000

...Program finished with exit code 0
Press ENTER to exit console.
```

- **Heap memory** for object data
- 2 different memory location for 2 different objects
- **Size of the object?**
  - Depends on the size of the number of variables and size of each variable
- **Who allocates size to the object?**
  - Constructor

Memory Allocation

```
class Computer:
    pass

c1=Computer()
c2=Computer()

print(id(c1))
print(id(c2))
```




```
140289883425760
140289883426000


...Program finished with exit code 0
Press ENTER to exit console.
```

- **Heap memory** for object data
- 2 different memory location for 2 different objects
- **Size of the object?**
  - Depends on the size of the number of variables and size of each variable
- **Who allocates size to the object?**
  - Constructor

Memory Allocation

```python
class Computer:

    def __init__(self):
        self.name = "NUV"
        self.age = 28

c1=Computer()
c2=Computer()

print(c1.name)
print(c2.name)
```

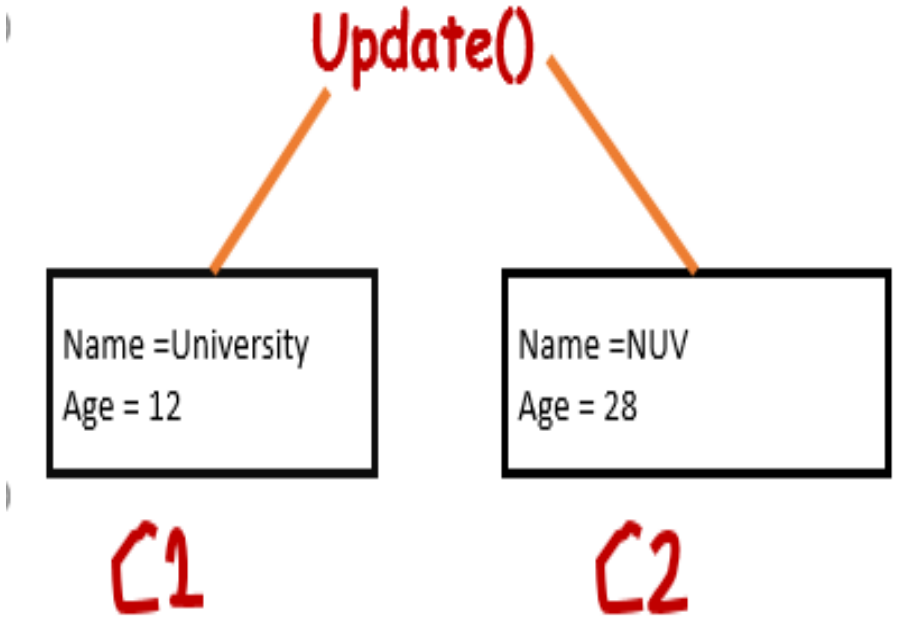
```
NUV
NUV


...Program finished with exit code 0
Press ENTER to exit console.
```

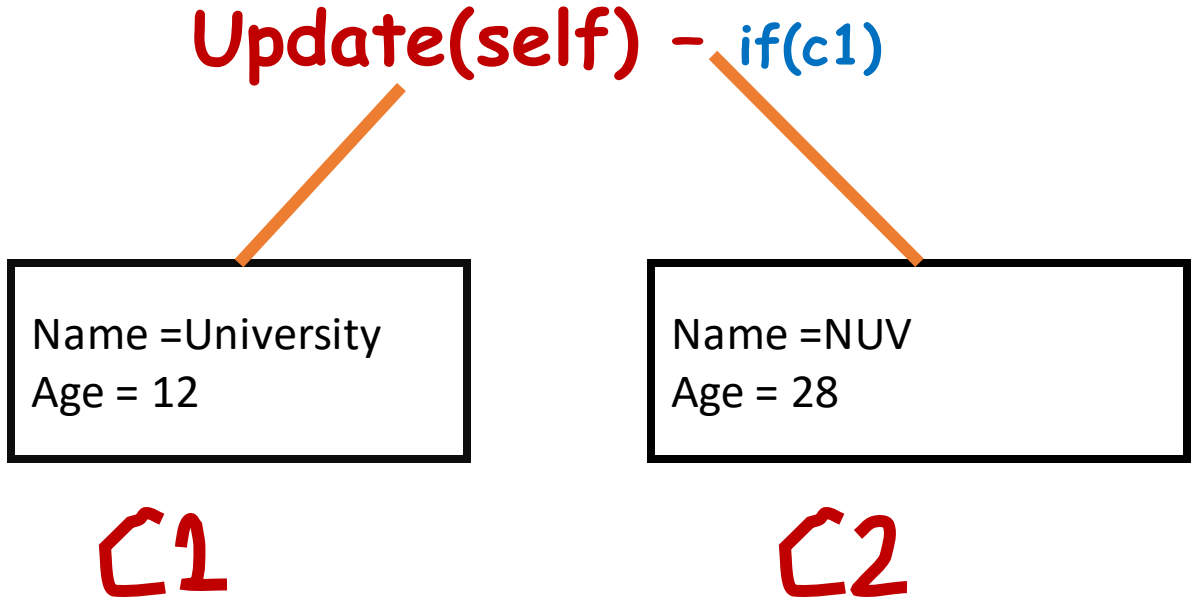
```python
class Computer:

    def __init__(self):
        self.name = "NUV"
        self.age = 28

c1=Computer()
c2=Computer()


c1.name = "University"
print(c1.name)
print(c2.name)
```

input

```
University
NUV


...Program finished with exit code
Press ENTER to exit console.
```

Memory Allocation





```python
class Computer:

    def __init__(self):
        self.name = "NUV"
        self.age = 28

    def update(self):
        self.age = 30

    def compare(self, ob):
        if self.age == ob.age:
            return True
        else:
            return False

c1=Computer()
c2=Computer()

if c1.compare(c2): # c1 - self, c2 - ob
    print("same")
else:
    print("different")

c1.update()

print(c1.name)
print(c2.name)
```

```
same
NUV
NUV


...Program finished with exit code 0
Press ENTER to exit console.
```

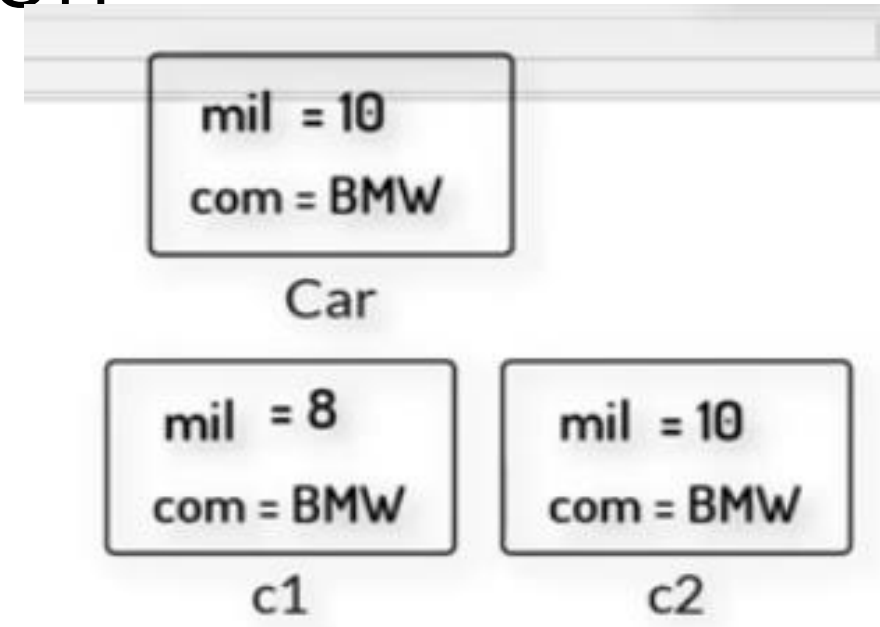
Types of Variable in Python

Types of Variable in Python

- 2 types of variable
  - Instance variable
  - Class (static) variable

Types of Variable in Python

```python
class Car:

    def __init__(self):
        self.mil = 10
        self.com = "BMW"

c1 = Car()
c2 = Car()

c1.mil = 8

print(c1.com, c1.mil)
print(c2.com, c2.mil)
```

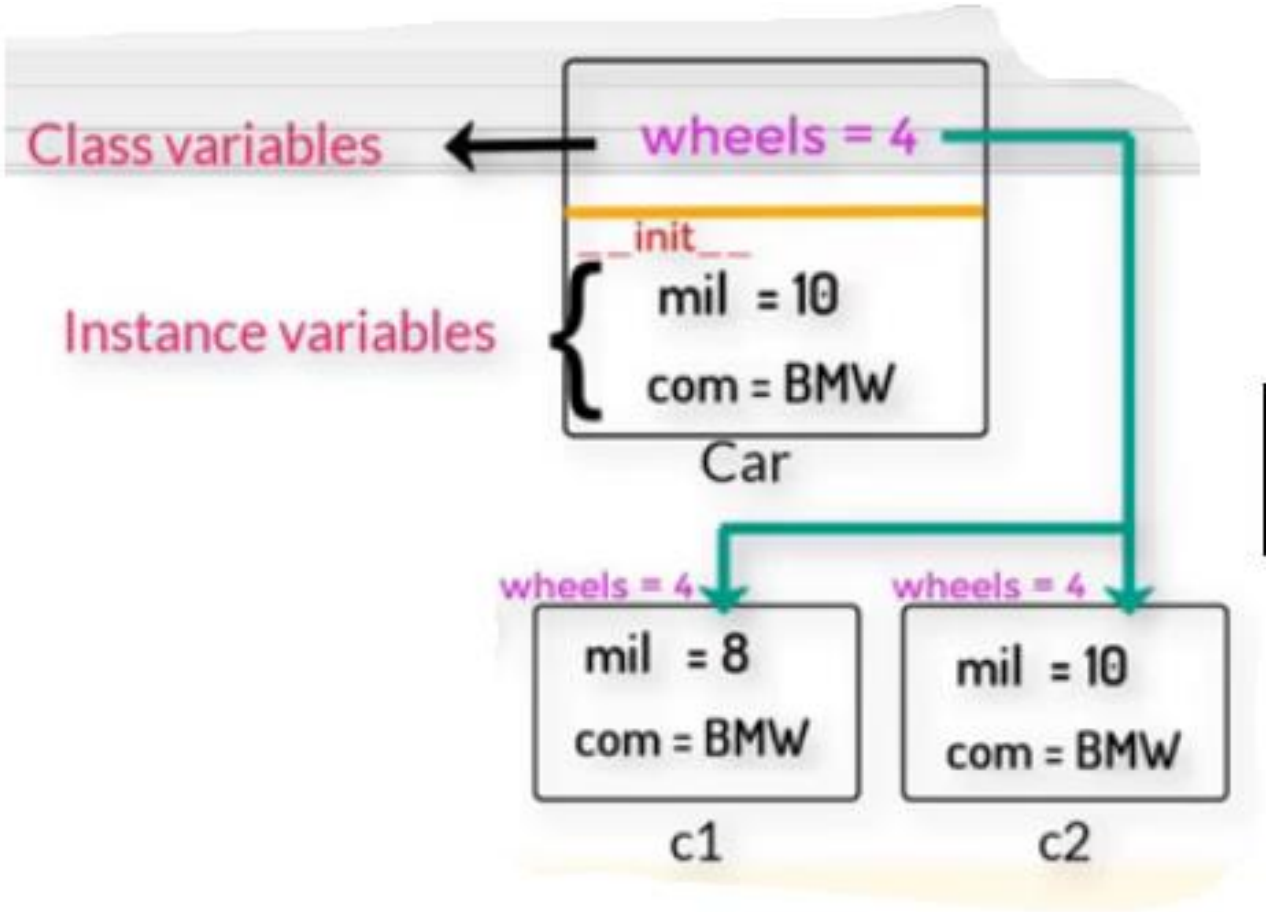


input

```
BMW 8
BMW 10


...Program finished with exit code 0
Press ENTER to exit console.
```

Types of Variable in Python

- **Namespace** is an area where you create and store object/variable
- **Class namespace**
  - Class variable also called **Static Variable**
- **Object/instance namespace**



Types of Variable in Python

```python
class Car:

    wheels = 4 #class name space

    def __init__(self):
        self.mil = 10       #instance namespace
        self.com = "BMW"    #instance namespace

c1 = Car()
c2 = Car()

c1.mil = 8

print(c1.com, c1.mil, c1.wheels)
print(c2.com, c2.mil, c2.wheels)
```

- To update value of class namespace need to write following:
- ***Car.wheel = 5***

input

```
BMW 8 4
BMW 10 4


...Program finished with exit code 0
Press ENTER to exit console.
```

```python
class Student:
    'A student class'
    stuCount = 0

    # initialization or constructor method of
    def __init__(self):

        # class Student
        self.name = input('enter student name:')
        self.rollno = input('enter student rollno:')
        Student.stuCount += 1

    # displayStudent method of class Student
    def displayStudent(self):
        print("Name:", self.name, "Rollno:", self.rollno)
```

```python
stu1 = Student()
stu2 = Student()
stu3 = Student()
stu1.displayStudent()
stu2.displayStudent()
stu3.displayStudent()
print('total no. of students:',
student.stuCount)
```

```python
class Student:
    'A student class'
    stuCount = 0

    # initialization or constructor method of
    def __init__(self):

        # class Student
        self.name = input('enter student name:')
        self.rollno = input('enter student rollno:')
        Student.stuCount += 1

    # displayStudent method of class Student
    def displayStudent(self):
        print("Name:", self.name, "Rollno:", self.rollno)
```

```python
stu1 = Student()
stu2 = Student()
stu3 = Student()
stu1.displayStudent()
stu2.displayStudent()
stu3.displayStudent()
print('total no. of students:', student.stuCount)
```

Types of Methods

Types of Methods

- 3 types
- Instance methods
- Class methods
- Static methods

Instance methods

```python
class Student:

    school = "NUV"

    def __init__(self, m1, m2, m3):
        self.m1 = m1
        self.m2 = m2
        self.m3 = m3

    def avg(self):
        return (self.m1+self.m2+self.m3)/3

s1 = Student(25,56,89)
s2 = Student(23,55,41)


print(s1.m1,s1.m2,s1.m3)
print(s1.avg())
print(s2.avg())
```

```
25 56 89
56.66666666666664
39.66666666666664


...Program finished with exit code 0
Press ENTER to exit console.
```

Instance methods

- The purpose of instance methods is to set or get details about instances (objects), and that is why they're known as instance methods. They are the most common type of methods used in a Python class.
- They have one default parameter- **self,** which points to an instance of the class. *Although you don't have to pass that every time.* You can change the name of this parameter but it is better to stick to the convention i.e **self.**
- Any method you create inside a class is an instance method unless you specially specify Python otherwise.

Class methods

```python
class Student:

    school = "NUV"

    def __init__(self, m1, m2, m3):
        self.m1 = m1
        self.m2 = m2
        self.m3 = m3

    def avg(self):
        return (self.m1+self.m2+self.m3)/3

    #@classmethod
    def info(cls):
        return cls.school

s1 = Student(25,56,89)
s2 = Student(23,55,41)


print(s1.avg())
print(s2.avg())
print(Student.info())
```

input

```
56.666666666666664
39.666666666666664
Traceback (most recent call last):
  File "main.py", line 23, in <module>
    print(Student.info())
TypeError: info() missing 1 required positional argument: 'cls'


...Program finished with exit code 1
Press ENTER to exit console.
```

Class methods

```python
class Student:

    school = "NUV"

    def __init__(self, m1, m2, m3):
        self.m1 = m1
        self.m2 = m2
        self.m3 = m3

    def avg(self):
        return (self.m1+self.m2+self.m3)/3

    @classmethod
    def info(cls):
        return cls.school

s1 = Student(25,56,89)
s2 = Student(23,55,41)


print(s1.avg())
print(s2.avg())
print(Student.info())
```

```
56.666666666666664
39.666666666666664
NUV


...Program finished with exit code 0
Press ENTER to exit console.
```

Class methods

- The purpose of the class methods is to set or get the details (status) of the class. That is why they are known as class methods. They can't access or modify specific instance data. They are bound to the class instead of their objects. Two important things about class methods:
- In order to define a class method, you have to specify that it is a class method with the help of the @classmethod decorator
- Class methods also take one default parameter- **cls,** which points to the class. Again, this not mandatory to name the default parameter "cls". But it is always better to go with the conventions

Class methods

```
class My_class:

  @classmethod
  def class_method(cls):
    return "This is a class method."

obj = My_class()
obj.class_method()
```

```
My_class.class_method()
```

- Without creating an instance of the class, you can call the class method with – **Class_name.Method_name()**

Static methods

```python
class Student:

    school = "NUV"

    def __init__(self, m1, m2, m3):
        self.m1 = m1
        self.m2 = m2
        self.m3 = m3

    def avg(self):
        return (self.m1+self.m2+self.m3)/3

    @staticmethod
    def info():
        print("Student Class")


s1 = Student(25,56,89)
s2 = Student(23,55,41)
```

- Nothing to do with class or instance variables
- ***Student.info()***

```
input
56.666666666666664
39.666666666666664
Student Class


...Program finished with exit code 0
Press ENTER to exit console.
```

Static methods

- Static methods cannot access the class data. In other words, they do not need to access the class data.
- They are self-sufficient and can work on their own.
- Since they are not attached to any class attribute, they cannot get or set the instance state or class state.
- In order to define a static method, we can use the @staticmethod decorator (in a similar way we used @classmethod decorator).
- Unlike instance methods and class methods, we do not need to pass any special or default parameters.

Types of Methods

- Here’s a summary of the explanation we’ve seen:
  - An instance method knows its instance (and from that, it’s class)
  - A class method knows its class
  - A static method doesn’t know its class or instance

When to Use Which Python Method?

- **Class Method –** The most common use of the class methods is for creating **factory methods**. Factory methods are those methods that return a class object (like a constructor) for different use cases.

```python
from datetime import date

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

# a class method to create a Dog object with birth year.
  @classmethod
  def Year(cls, name, year):
    return cls(name, date.today().year - year)
```

```python
Dog1 = Dog('Bruno', 1)
Dog1.name , Dog1.age
```

```
('Bruno', 1)
```

```python
Dog2 = Dog.Year('Dobby', 2017)
Dog2.name , Dog2.age
```

```
('Dobby', 3)
```

- **Static Method –** They are used for creating utility functions. For accomplishing routine programming tasks we use utility functions.

```
class Calculator:

  @staticmethod
  def add(x, y):
    return x + y

print('The sum is:', Calculator.add(15, 10))
```

Practice Questions

1. Write a Python class which has two methods get_String and print_String. get_String accept a string from the user and print_String print the string in upper case.
2. Write a **Rectangle class** in Python language, allowing you to build a rectangle with **length** and **width** attributes.
   - Create a **Perimeter()** method to calculate the perimeter of the rectangle and a **Area()** method to calculate the area of the rectangle.
   - Create a method **display()** that display the length, width, perimeter and area of an object created using an instantiation on rectangle class.
3. Write a Python class to implement pow(x, n).
4. Write a Python class to reverse a string word by word

```
class IOString:
    def __init__(self):
        self.str1 = ""

    def get_String(self):
        self.str1 = input()

    def print_String(self):
        print(self.str1.upper())

str1 = IOString()
str1.get_String()
str1.print_String()
```

```
class py_solution:
  def pow(self, x, n):
      if x==0 or x==1 or n==1:
         return x

      if x==-1:
         if n%2 ==0:
            return 1
         else:
            return -1
```

```
if n==0:
         return 1
      if n<0:
         return 1/self.pow(x,-n)
      val = self.pow(x,n//2)
      if n%2 ==0:
         return val*val
      return val*val*x

print(py_solution().pow(2, -3));
print(py_solution().pow(3, 5));
print(py_solution().pow(100, 0));
```

```
class py_solution:
    def reverse_words(self, s):
        return ' '.join(reversed(s.split()))


print(py_solution().reverse_words('hello .py'))
```