What is difference between constructor and instance method?

Constructor	Instance Method
Constructor is a special method which is used for defining initial properties of object	Instance method is used for performing setter and getter operation
This method is executed automatically whenever object of class is created	This method is called explicitly, to perform setter and getter
Inside class one constructor is defined	Inside class more than one instance method is defined

Example:

```
class Student:
    def __init__(self):
        self.rollno=None
        self.name=None
    def print_student(self):
        print(self.rollno,self.name)


stud1=Student()
stud1.print_student()
stud2=Student()
stud2.print_student()
dict1={1:10,2:20}
dict1.get(1)
```

Output:

```
None None
None None
```

Setter operation is an operation which modifies or updates values of object. Getter operation is an operation which does not modifies or update values object.

Example:

```
class Triangle:
```

```
    def __init__(self):
        self.base=0.0
        self.height=0.0
    def update_base(self,b):
        self.base=b
    def update_height(self,h):
        self.height=h
    def find_area(self):
        a=self.base*self.height*0.5
        return a


t1=Triangle()
t1.update_base(1.5)
t1.update_height(1.2)
area1=t1.find_area()
t1.update_base(2.0)
area2=t1.find_area()
print(f'{area1:.2f}')
print(f'{area2:.2f}'
```

Output:
0.90
1.20

Instance methods are used to operate instance variables.

The members of the class can be declared or defined as,

1. Private
2. Protected
3. Public

Private, protected and public are called access modifiers or specifiers.
Access modifiers define accessibility of members of the class.
Default members of class are public, these members are accessible within class and outside the class.

Example:

```
class A:
```

```
    def __init__(self):
        self.x=100 # public variable
    def m1(self): # public method
        print("public m1 method")


obj1=A()
obj1.m1()
print(obj1.x)
```

Output:
public m1 method
100

Private
Private members of the class are prefix with __
Private members are accessible within class but cannot accessible outside the class.
In object oriented programming data hiding achieved by declaring data members or variables are private.

Example:

```
class A:
    def __m1(self):
        print("private method")
    def __init__(self):
        self.__x=100 # private


obj1=A()
#obj1.__m1()
print(obj1.__x)
```

Output:
Error

Private members are accessible within class but cannot accessible outside the class. In order to access private members outside the class we need to use public methods.

Protected

Any member of class is prefix with _ is called protected members. Protected members are accessible within class and in inherited class but cannot accessible outside the class.

	Within class	Within inherited class	Outside class
Private (__)	YES	NO	NO
Protected (_)	YES	YES	NO
Public	YES	YES	YES

Example:

```
class A:
    def __m1(self):
        print("private method")
    def m2(self):
        print("public method")
        self.__m1()

obj1=A()
obj1.m2()
#obj1.__m1()
```

Output:

public method
private method

Example:

```
class Account:
    def __init__(self):
        self.__accno=None # private
        self.__name=None # private
        self.__balance=None # private
    def input_account(self):
        self.__accno=int(input("AccountNo "))
        self.__name=input("Name ")
        self.__balance=float(input("Balance "))
```

```
    def print_account(self):
        print(f'AccountNo {self.__accno}')
        print(f'CustomerName {self.__name}')
        print(f'Balance {self.__balance}')


acc1=Account()
acc1.input_account()
acc1.print_account()
```

Output:

```
AccountNo 1
Name suresh
Balance 56000
AccountNo 1
CustomerName suresh
Balance 56000.0
```

Example:

```
class Stack:
    def __init__(self):
        self.__s=list()
    def push(self,value):
        self.__s.append(value)
    def pop(self):
        if len(self.__s)==0:
            print("Stack is empty")
        else:
            return self.__s.pop()


stack1=Stack()
stack1.push(10)
stack1.push(20)
a=stack1.pop()
print(a)
b=stack1.pop()
print(b)
```

```
c=stack1.pop()
print(c)
```

Output

```
20
10
Stack is empty
None
```