Special methods in python

Special methods in python

- Python classes have a number of special methods.
- These methods have a double leading underscore and a double trailing underscore in their names. You can informally refer to them as dunder methods because of the double underscores in their names.
- Some times they can be called magic methods.

Example

```
class Person:
    def __init__(self, first_name, last_name, age):
        self.first_name = first_name
        self.last_name = last_name
        self.age = age
person = Person('John', 'Doe', 25)
print(person)
```

Output:

```
<__main__.Person object at 0x0000023CA16D13A0>
```

- When you use the print() function to display the instance of the Person class, the print() function shows the memory address of that instance.
- Sometimes, it’s useful to have a string representation of an instance of a class.
- To customize the string representation of a class instance, the class needs to implement the __str__ magic method.
- Internally, Python will call the __str__ method automatically when an instance calls the str() method.
- **Note that** the print() function converts all non-keyword arguments to strings by passing them to the str() before displaying the string values.
- __str__ method to customize the string representation of an instance of a class.

Example

```
class Person:
    def __init__(self, first_name, last_name, age):
        self.first_name = first_name
        self.last_name = last_name
        self.age = age

    def __str__(self):
        return str(self.first_name)+'\t'+str(self.last_name)+'\n'+str(self.age)
person = Person('John', 'Doe', 25)
print(person)
```

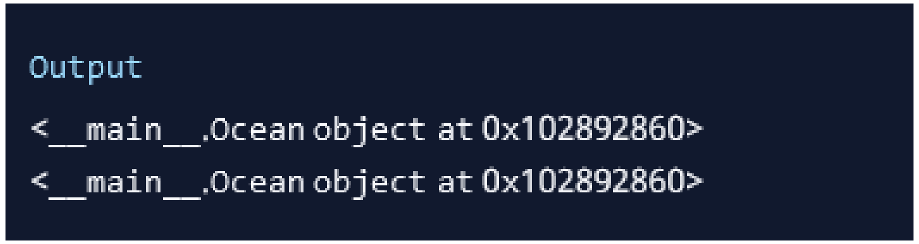
The output shows that

the str() function calls __str__() and returns a human-friendly string,

repr() function calls __repr__() and returns a more information-rich

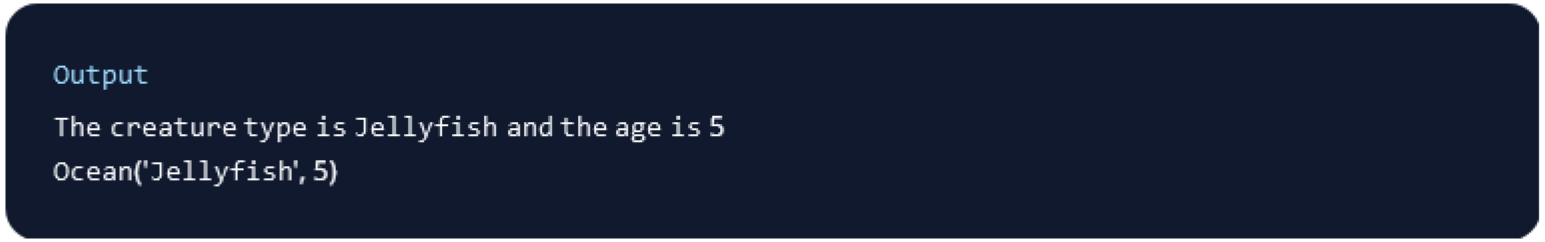
string that can be used to recreate the object.

```
class Ocean:

    def __init__(self, sea_creature_name, sea_creature_age):
        self.name = sea_creature_name
        self.age = sea_creature_age

c = Ocean('Jellyfish', 5)

print(str(c))
print(repr(c))
```

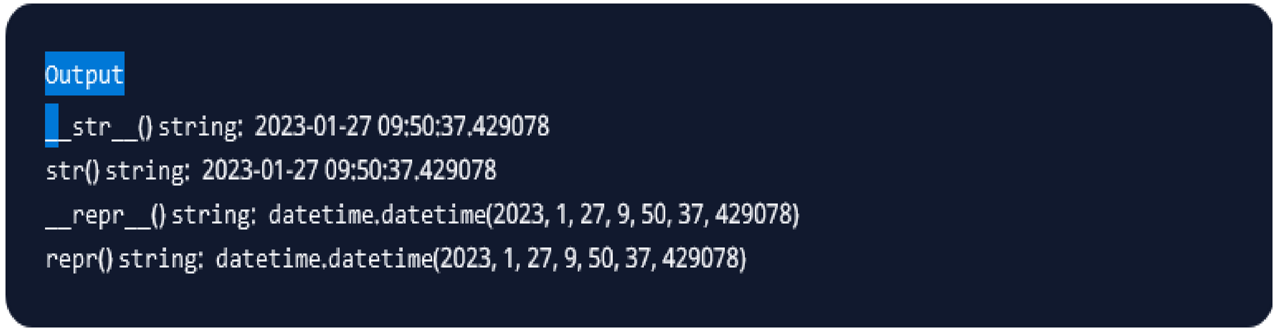


```
Output
<__main__.Ocean object at 0x102892860>
<__main__.Ocean object at 0x102892860>
```

```
class Ocean:

  def __init__(self, sea_creature_name, sea_creature_age):
    self.name = sea_creature_name
    self.age = sea_creature_age

  def __str__(self):
    return f'The creature type is {self.name} and the age is {self.age}'

  def __repr__(self):
    return f'Ocean(\'{self.name}\', {self.age})'

c = Ocean('Jellyfish', 5)

print(str(c))
print(repr(c))
```


```
Output
The creature type is Jellyfish and the age is 5
Ocean('Jellyfish', 5)
```

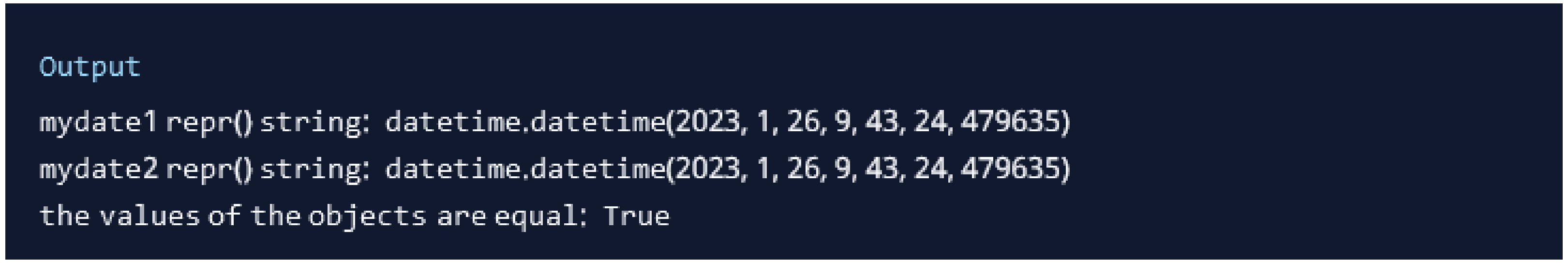
```
from datetime import datetime

print(repr(datetime.now()))  # datetime.datetime(2024, 1, 31, 12, 34, 56, 789012)
print(str(datetime.now()))   # 2024-01-31 12:34:56.789012



import datetime
mydate = datetime.datetime.now()
print("__str__() string: \n",mydate.__str__())
print("str() string: \n", str(mydate))

print("__repr__() string: ", mydate.__repr__())
print("repr() string: ", repr(mydate))
```



```
Output
__str__() string:  2023-01-27 09:50:37.429078
str() string:  2023-01-27 09:50:37.429078
__repr__() string:  datetime.datetime(2023, 1, 27, 9, 50, 37, 429078)
repr() string:  datetime.datetime(2023, 1, 27, 9, 50, 37, 429078)
```

you can use the repr() function with the eval() function to create a new object from the string:

```
import datetime
mydate1 = datetime.datetime.now()
mydate2 = eval(repr(mydate1))
print("mydate1 repr() string: ", repr(mydate1))
print("mydate2 repr() string: ", repr(mydate2))
print("the values of the objects are equal: ", mydate1==mydate2)
```



```
Output
mydate1 repr() string:  datetime.datetime(2023, 1, 26, 9, 43, 24, 479635)
mydate2 repr() string:  datetime.datetime(2023, 1, 26, 9, 43, 24, 479635)
the values of the objects are equal:  True
```

Instance variable Vs. Class variable

- **Instance variables**: If the value of a variable varies from object to object, then such variables are called instance variables.
- **Class Variables or Static Variables**: A class variable is a variable that is declared inside of class, but outside of any instance method or __init__() method.

```
class Student:
                # Class variable
  school_name = 'Myanamr Institute of Information Technology‘
                #instance variable
  def __init__(self, name, roll_no):
    self.name = name
    self.roll_no = roll_no
  def __str__(self):
     return str(self.name)+'\n'+str(self.roll_no)+'\n'+str(Student.school_name)

# create first object
s1 = Student('Aye Aye', 10)
print(s1)
#print(s1.name, s1.roll_no, Student.school_name)
# access class variable

# create second object
s2 = Student('Latt Latt', 20)
# access class variable
#print(s2.name, s2.roll_no, Student.school_name)
print(s2)
```

```
class Student:
   #this is a class variable or static variables
   n=10

print(Student.n)
Student.n+=1
print(Student.n)
```

Accessing instance and class variable

```
class Student:
    # Class variable
    school_name = 'ABC School '

    # constructor
    def __init__(self, name, roll_no):
        self.name = name
        self.roll_no = roll_no

    # Instance method
    def show(self):
        print('Inside instance method')
        # access using self
        print(self.name, self.roll_no, self.school_name)
        print(Student.school_name)

# create Object
s1 = Student('Emma', 10)
s1.show()
print('Outside class')
# access class variable outside class
# access using object reference
print(s1.school_name)
# access using class name
print(Student.school_name)
```

```
#Python program to understand instance variable
class Sample:
#instance vars sample
    def __init__(self):
        self.x=10
#This is and instance method
    def modify(self):
        self.x+=10
#create 2 instances
s1=Sample()
s2=Sample()
print('-----------')
#print('before modify',s1.x)
#print('before modify',s2.x)
#modify in s1
s1.modify()
print('-----------')
print('-----------')
print('-----------',s1.x)
print('-----------',s2.x)
```

```
class Student:
    #this is a class var
    n=10
s1=Student()
print('test for class var s1',s1.n)
s2=Student()
print('test for class var s2',s2.n)
'''
class Student:
    #this is a class var
    n=10
s1=Student()
#print('test for class var s1',s1.n)
s1.n+=1
print('test for class var s1',s1.n)
s2=Student()
print(s2.n)
```

Printing modules

```
# Printing module names using the name attribute __name__

import turtle as t
import math as m
import timeit as ti

print(__name__) # __name__ without the module prints that this is the main module
print(t.__name__) # Print turtle
print(m.__name__) # Print math
print(ti.__name__) # Print timeit
```

References

- https://pynative.com/python-instance-variables/
- https://realpython.com/instance-class-and-static-methods-demystified/
- Ref:https://www.digitalocean.com/community/tutorials/python-str-repr-functions

- https://pynative.com/python-class-method-vs-static-method-vs-instance-method/
- https://pynative.com/python-static-method/
- https://pynative.com/python-class-method/