UNIVERSITY OF ENGINEERING AND TECHNOLOGY PESHAWAR, JALOZAI CAMPUS

Lab 6: str and repr and docstring

Lab Title: EE-271 "OOP & Data Structures Lab"

Time: 10 min/ Task

Lab report work

1. Consider the following code.

```
import math
class Point:
    'Represents a point in two-dimensional geometric coordinates'
    def __init__(self, x=0, y=0):
        '''Initialize the position of a new point. The x and y
        coordinates can be specified. If they are not, the
        point defaults to the origin.'''
        self.move(x, y)
    def move(self, x, y):
        '''Move the point to a new location in 2D space.'''
        self.x = x
        self.y = y
    def reset(self):
        '''Reset the point back to the geometric origin: 0, 0'''
        self.move(0, 0)
    def __repr__(self) -> str:
        return f"{type(self).__name__}(x={self.x}, y={self.y})"
    def __str__(self) -> str:
        return f"{type(self).__name__}(x={self.x}, y={self.y})"
    def calculate_distance(self, other_point):
        """Calculate the distance from this point to a second
        point passed as a parameter.
        This function uses the Pythagorean Theorem to calculate
        the distance between the two points. The distance is
        returned as a float."""
        return math.sqrt((self.x - other_point.x)**2 +(self.y -
other_point.y)**2)
```

   a. Define an object point = Point(3, 9) and print its coordinate.
   b. Print the class docstring.
      - Hints: Use the object name to access the class-level docstring by using __doc__. (point.__doc__)
   c. Print different method docstring using object.meteod. __doc__
   d. Add repr and str methods.
   e. Define a second object and test the str and rpr methods.
   f. Also print the different docstrings using __doc__.
   g. Use the help method to print the class details.

Lab task

Motivation:

2. Consider the following class definition from the last class.

```
class Dog:
  species = "Canis familiaris"
  def __init__(self, name, age):
    self.name = name
    self.age = age
  # Instance method
  def description(self):
    return f"{self.name} is {self.age} years old"
  # Another instance method
  def speak(self, sound):
    return f"{self.name} says {sound}"
```

a. Run the code: miles.description()

b. Run the code: print(miles)

c. Copy this list to the code editor, names = ["Fletcher", "David", "Dan"]. Now run print(names). Note: What is the difference between the two prints, the first one prints a memory address while the second one prints the list. The second one is good and attractive.

d. Now modify the dog class as follows:

```
class Dog:
  species = "Canis familiaris"
  def __init__(self, name, age):
    self.name = name
    self.age = age
  # Instance method
  def __str__(self):
    return f"{self.name} is {self.age} years old"
  # Another instance method
  def speak(self, sound):
    return f"{self.name} says {sound}"
```

In the above code, only the description function is replaced with a new name __str__. Now make an object/instance of the Dog class and then run the code print ().

Hints:

i. The. description() method defined in the above Dog class returns a string containing information about the Dog instance miles. When writing your own classes, it is a good idea to have a method that returns a string containing useful information about an instance of the class.

3. Consider the car class.

```
class Car:
    def __init__(self, color, mileage):
        self.color = color
        self.mileage = mileage
```

a. Define an object:

my_car = Car('red', 37281)

b. Run: `print(my_car)`
c. Run: `my_car`

4. Modify the car class by introducing an __str__ method.

```
class Car:
    def __init__(self, color, mileage):
        self.color = color
        self.mileage = mileage
    def __str__(self):
        return f"The {self.color} car has {self.mileage:,} miles"
```

a. Define an object: `my_car = Car('red', 37281)`
b. Run: `print(my_car.color, my_car.mileage)`
c. Run: `print(my_car)`
d. Run: `my_car`
e. Add docstrings for the class and functions.
f. Print the docstring using __doc__.
g. Use the help function to print the class details.

Note:

i. _str_ is one of Python's "dunder" (double-underscore) methods and gets called when you try to convert an object into a string.

5. Complete car class

```
class Car:
  def __init__(self, color, mileage):
    self.color = color
    self.mileage = mileage
  def __repr__(self):
    return (f'{self.__class__.__name__}('
    f'{self.color!r}, {self.mileage!r})')
  def __str__(self):
    return f'a {self.color} car'
```

a. Define an object: `my_car = Car('red', 37281)`
b. Run: `print(my_car.color, my_car.mileage)`
c. Run: `print(my_car)`
d. Run: `my_car`

Note:

i. _class_._name_ attribute, which will always reflect the class' name as a string.

6. Consider the car class again. Add a new dunder method __repr__.

```
class Car:
      def __init__(self, color, mileage):
      self.color = color
      self.mileage = mileage
      def __repr__(self):
            return '__repr__ for Car'
      def __str__(self):
            return '__str__ for Car'
```

a. Define an object: `my_car = Car('red', 37281)`

b. Run: `print (my_car.color, my_car.mileage)`
c. Run: `print(my_car)`
d. Run: `my_car`

7. Consider the following code.

```
import math
class Point:
    'Represents a point in two-dimensional geometric coordinates'
    def __init__(self, x=0, y=0):
        '''Initialize the position of a new point. The x and y
        coordinates can be specified. If they are not, the
        point defaults to the origin.'''
        self.move(x, y)
    def move(self, x, y):
        '''Move the point to a new location in 2D space.'''
        self.x = x
        self.y = y
    def reset(self):
        '''Reset the point back to the geometric origin: 0, 0'''
        self.move(0, 0)
    def calculate_distance(self, other_point):
        """Calculate the distance from this point to a second
        point passed as a parameter.
        This function uses the Pythagorean Theorem to calculate
        the distance between the two points. The distance is
        returned as a float."""
        return math.sqrt((self.x - other_point.x)**2 +(self.y -
other_point.y)**2)
```

h. Define an object point = Point(3, 9) and print its coordinate.
i. Print the class docstring.
   - Hints: Use the object name to access the class-level docstring by using __doc__. (point.__doc__)
j. Print different method docstring using object.meteod. __doc__

1. Point class

```
# point.py

class Point:
    def __init__(self, x, y):
        print("Initialize the new instance of Point.")
        self.x = x
        self.y = y

    def __repr__(self) -> str:
        return f"{type(self).__name__}(x={self.x}, y={self.y})"
```

a. Run the following code

```
point = Point(21, 42)
point
```

b. The __init__ can be called by the instance and pass value directly. Run the following code.

```
point.__init__(34, 45)
point.x
point.y
point
```

Note:

- **Docstring** is important to write API documentation that clearly summarizes what each object and method does.
- Often, docstrings are quite long and span multiple lines (the style guide suggests that the line length should not exceed 80 characters), which can be formatted as multi-line strings, enclosed in matching triple apostrophe (''') or triple quote (""") characters.
- A docstring should clearly and concisely summarize the purpose of the class or method it is describing. It should explain any parameters whose usage is not immediately obvious and is also a good place to include short examples of how to use the API.
- What is API? https://www.youtube.com/watch?v=tI8ijLpZaHk