

CSC148-Section:L0301

Week#1-Wednesday

Instructed by

AbdulAziz Al-Helali

a.alhelali@mail.utoronto.ca

Office hours: Wednesday 11-1, BA2230.

Slides adapted from Professor Danny Heap and Jacqueline Smith slides winter17

Announcements

- Tutorial rooms will be posted later today
 - If you are registered late check with BA4208

Outline

- Continue Point class
- Build class Point. . .
in that **deeply wrong** way
- Build Rational class

Cont. Exercise: building Point class

Somewhere in the real world there is a description of points in two-dimensional space:

In two dimensions, a point is two numbers (coordinates) that are treated collectively as a single object. Points are often written in parentheses with a comma separating the coordinates. For example, $(0, 0)$ represents the origin, and (x, y) represents the point x units to the right and y units up from the origin. Some of the typical operations that one associates with points might be calculating the distance of a point from the origin, or from another point, or finding a midpoint of two points, or asking if a point falls within a given rectangle or circle.

Define a class API:

1. choose a **class name** and write a brief **description** in the class docstring.
2. write **some examples** of client code that uses your class
3. decide what **services** your class should provide as public methods, for each method declare an API (examples, header, type contract, description)
4. decide which **attributes** your class should provide without calling a method, list them in the class docstring

Implement the class:

1. body of special methods `__init__`, `__eq__`, and `__str__`
2. body of other methods
3. testing (more on this later)

point class implementation

```
"""  
point module  
"""  
from typing import Any  
  
class Point:  
    """ Represent a two-dimensional point  
  
    x - horizontal position  
    y - vertical position  
    """  
    x: float  
    y: float  
  
    def __init__(self, x: float, y: float) -> None:  
        """ Initialize a new point  
        """  
        self.x, self.y = float(x), float(y)
```



```
def __eq__(self, other: Any) -> bool:
    """ Return whether self is equivalent to other.
```

```
>>> Point(3, 5) == Point(3.0, 5.0)
```

```
True
```

```
>>> Point(3, 5) == Point(5, 3)
```

```
False
```

```
"""
```

```
return (type(self) == type(other)
        and self.x == other.x
        and self.y == other.y)
```

In __eq__ method:

- 1- compare the **types** of objects
- 2- Compare **all attributes** in that object

```
def __str__(self) -> str:
    """ Return a string representation of self
```

```
>>> print(Point(3, 5))
```

```
(3.0, 5.0)
```

```
"""
```

```
return "({}, {})".format(self.x, self.y)
```

In __str__ method:

Format the output string to be **exactly** the same as the **examples** otherwise it will fail testing, notice the space before 5.0


```
def distance_from_origin(self) -> float:
    """ Return the distance from the origin of this point

    >>> Point(3, 4).distance_from_origin()
    5.0
    """
    return (self.x**2 + self.y**2)**(1/2)
```

```
if __name__ == "__main__":
    from doctest import testmod
    testmod()
```

self attribute

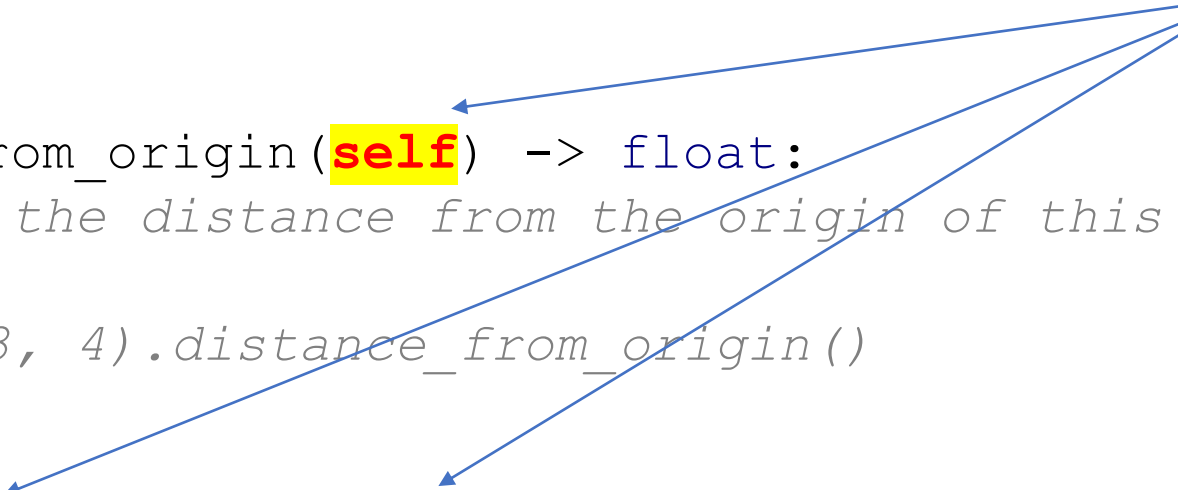
- In the `__init__` **method**:
 - **self** refers to the newly created instance or object
- in other **class methods**,
 - it refers to the object whose **method** was called.

self can be replaced with any other name like **this** or any name you like but in Python, it is a good convention to use **self**

Example:

```
def distance_from_origin(self) -> float:
    """ Return the distance from the origin of this point

    >>> Point(3, 4).distance_from_origin()
    5.0
    """
    return (self.x ** 2 + self.y ** 2) ** 0.5
```



The **Any** Type

“A special kind of type is Any. A static type checker will treat every type as being compatible with Any and Any as being compatible with every type.”[1]

To use this type you must import it and the **beginning of your class**

```
from typing import Any
```

Example:

```
def __eq__(self, other: Any) -> bool:
    """ Return whether self is equivalent to other.
    >>> Point(3, 5) == Point(3.0, 5.0)
    True
    >>> Point(3, 5) == Point(5, 3)
    False
    """
    return (type(self) == type(other) and
            self.x == other.x and self.y == other.y)
```

Long Lines

- In CSC148, we follow pep 8, and not CSC108, style in preferring **to use parentheses** for long lines.

Example:

```
def __eq__(self, other: Any) -> bool:
    """ Return whether self is equivalent to other.

    >>> Point(3, 5) == Point(3.0, 5.0)
    True
    >>> Point(3, 5) == Point(5, 3)
    False
    """

    return (type(self) == type(other)
            and self.x == other.x
            and self.y == other.y)
```

Weird things

- what happens if, after declaring Point, you try
 `print(Point.x)`
 OR
 `Point.y = 17`
- methods can be invoked in two equivalent ways:
 `p = Point(3, 4)`
 `p.distance_to_origin()`
 5.0
 `Point.distance_to_origin(p)`

in each case the first parameter, conventionally self, refers to the **instance named p**

build class Point. . .

Do not use it

in that **deeply wrong**, but informative, way

```
>>> class Point:
...     pass
...
>>> def initialize(point, x, y):
...     point.x = x
...     point.y = y
...
>>> def distance(point):
...     return (point.x**2 + point.y**2) ** (1 / 2)
...
```

```
>>> Point.__init__ = initialize
>>> Point.distance = distance
>>> p2 = Point(12, 5)
>>> p2.distance()
13.0
>>>
```

Exercise: build Rational class

Here is a description of rational numbers, the fractions we learned in grade school:

Rational numbers are ratios of two integers p/q , where p is called the numerator and q is called the denominator. The denominator q is non-zero. Operations on rationals include addition, multiplication, and comparisons: $>$, $<$, \geq , \leq , $=$.



http://www.teach.cs.toronto.edu/~csc148h/winter/lecturedata/Danny/W1/rational_exercise.pdf

Computer Science

UNIVERSITY OF TORONTO

Exercise: build Rational class

Here is a description of rational numbers, the fractions we learned in grade school:

Rational numbers are ratios of two integers p/q , where p is called the numerator and q is called the denominator. The denominator q is non-zero. Operations on rationals include addition, multiplication, and comparisons: $>$, $<$, \geq , \leq , $=$.

Define a class API:

1. choose a **class name** and write a brief **description** in the class docstring.
2. write **some examples** of client code that uses your class
3. decide what **services** your class should provide as public methods, for each method declare an API (examples, header, type contract, description)
4. decide which **attributes** your class should provide without calling a method, list them in the class docstring

Implement the class:

1. body of special methods `__init__`, `__eq__`, and `__str__`
2. body of other methods
3. testing (more on this later)

http://www.teach.cs.toronto.edu/~csc148h/winter/lecturedata/Danny/W1/rational_exercise.pdf