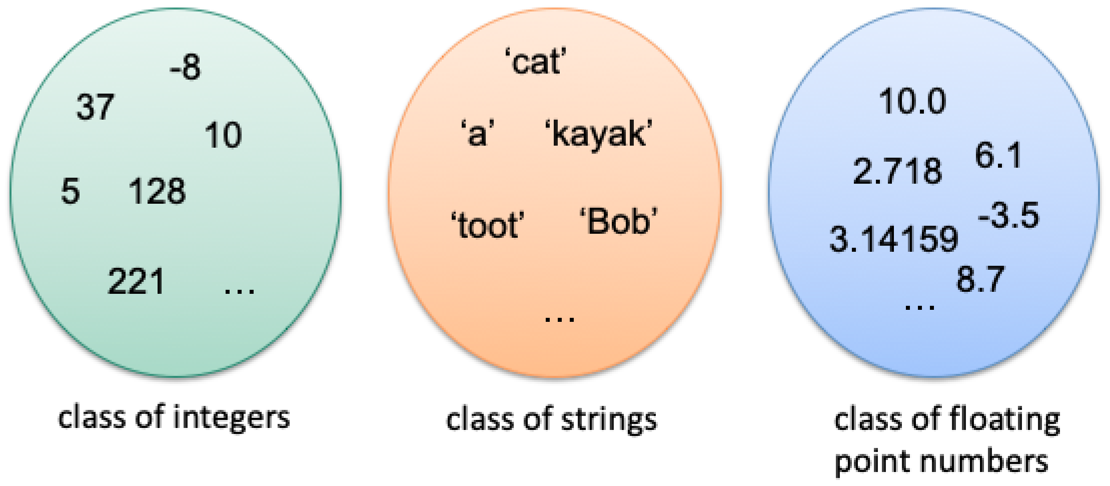
Class

A *class* is another name for a *type* in Python. A class defines a data type . A class definition characterizes the properties and the behavior of a group of objects that are of a certain kind.

We have been working with several built-in types such as str, int, float, list, tuple etc. For example an int type in Python defines how integer objects look like and how they behave, i.e., what operations can be performed on integer objects.



It is convenient to be able to define new classes to represent things with similar properties and behaviour, and manipulate these representations in your code. Object-oriented programming, which is based on the concepts of objects and classes, is one of the major programming paradigms. Its success lies in the fact that it makes it easier to design code in a modular way, re-use, maintain, and extend code over long periods of time, especially for large software projects.

For example, in a program that deals with students, courses, exams, etc (think of Beartracks), it will be convenient to have a class for students that characterizes their important properties (name, id, etc.) and behaviour (register for classes, drop classes, etc.).

Object

Objects are instances of a class.

- All objects *instantiated/created* from a certain class will have the same kind of properties, called *(instance) attributes*, which are defined in the class, but their values could be different.
- All objects of a certain class can be processed/manipulated by methods of that class.

- A new object of a class is created by calling a function that has the same name as the class. This is true already for the built-in types. The functions we have so far called "type conversion" functions are truly constructors that create a new object when called. Try, for instance: list() or str() and you will see that they create a new object of that type.

Instance Attribute

An instance attribute defined in a class is a variable/identifier that refers to a data object that characterises a property of an object of that class. Instance attributes are also sometimes called the member data of the class.

For example, an instance attribute of an object of type Circle could be the radius of the circle or an instance attribute of an object of type Person could be the name of the person. Instance attributes are referred to using the dot notation.
For instance, if the variable `person1` is bound to an object of type Person then we can refer to the `name` instance attribute of the object by the following expression:

```
person1.name
```

Similarly, if the variable `self` is bound to an object of type Circle then we can refer to the `radius` instance attribute of the object by the following expression:

```
self.radius
```

Method

A method is a function defined inside a class. A method implements the behavior and/or operations that can be performed on objects of the class. The first parameter in the method definition (called *self* by convention) is used to refer to the object on which the method is called upon.
We have called methods (e.g., lower and splitlines) on string objects in expressions such as:

```
answer.lower()
```

or

```
content.splitlines()
```

What goes in a class definition?

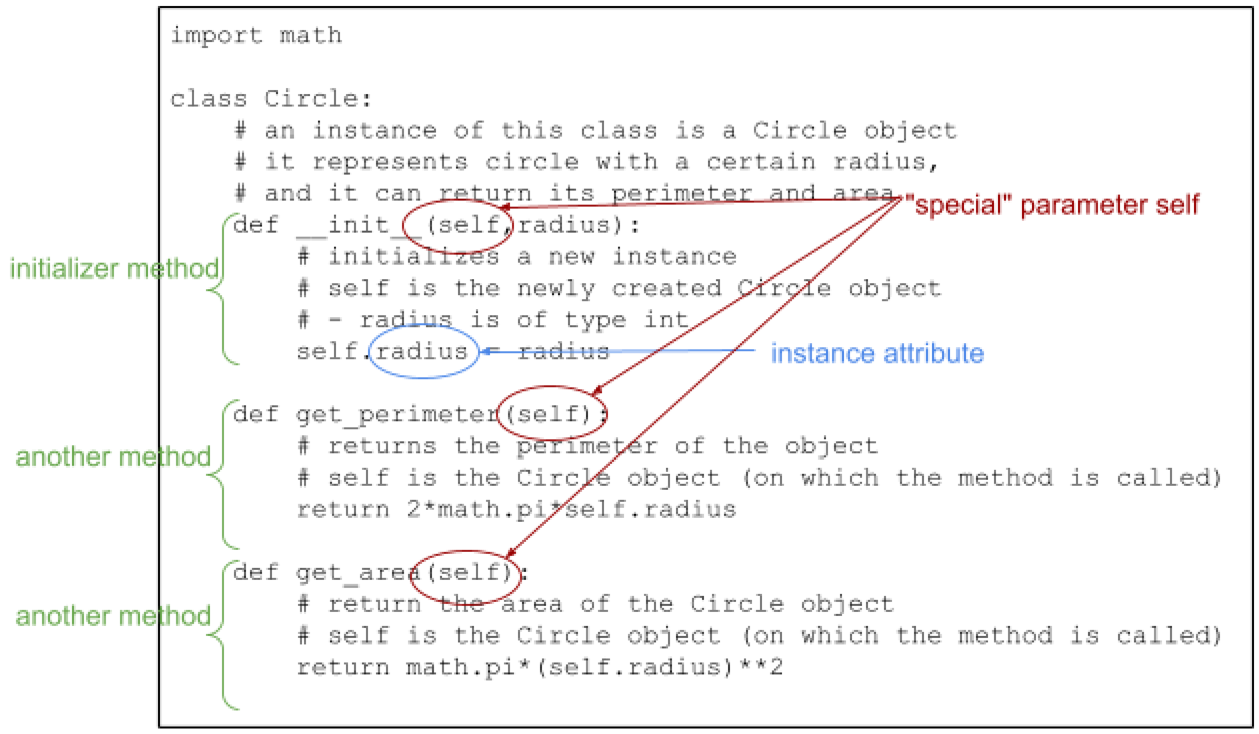
A class is usually made up of:

- An *initializer* method called __init__ that is used to initialize the instance attributes of a newly created object of the class. Possible arguments to the initializer have to be given to the function call that creates the object; they will be passed along to the initializer of the class.
- Other methods that implement the behavior of the object.

Class Definition Syntax

```
class <name of class>:
    def __init__(self,<parameter1>,<parameter2>,..):
        <method body>
    def <method_name>(self,<parameter1>,<parameter2>,..):
         <method body>
    ...
```

Class Definition Example



```
import math

class Circle:
    # an instance of this class is a Circle object
    # it represents circle with a certain radius,
    # and it can return its perimeter and area
    def __init__(self,radius):
        # initializes a new instance
        # self is the newly created Circle object
        # - radius is of type int
        self.radius = radius

    def get_perimeter(self):
        # returns the perimeter of the object
        # self is the Circle object (on which the method is called)
        return 2*math.pi*self.radius

    def get_area(self):
        # return the area of the Circle object
        # self is the Circle object (on which the method is called)
        return math.pi*(self.radius)**2
```

Using the Circle class

Assuming the Circle class exists we can write the following code to instantiate the Circle class:

`c1 = Circle(2)`

Explanation: A class is a callable in Python (like a function). When we call the Circle function with argument 2 , the __init__ method in the Circle class is also called, after the object has been created. At the time of this call, any argument given to the Circle function is passed to the __init__ method in the Circle class. The parameter self in the __init__ method definition will be bound to the object that c1 is bound to and the parameter radius in the __init__ method definition will be bound to the same object that the argument in the Circle function call is bound to which is the integer object 2. The instance attribute radius referred to as self.radius inside the body of the __init__ method is initialized with the value of 2.

Once the object c1 has been created we can call the method get_area() on that object:

`area_of_circle = c1.get_area()`

Explanation: When the method is called, the parameter self in the method definition of get_area will be bound to the Circle object c1.The method get_area uses the instance attribute self.radius to compute and return the area of the circle c1.

We can also call the method get_perimeter on the Circle object c1.

`perimeter_of_circle = c1.get_perimeter()`

Explanation: When the method is called, the parameter self in the method definition of get_area is bound to the Circle object c1.The method get_perimeter uses the instance attribute self.radius to compute and return the area of c1.

We can create multiple circle objects and use the methods defined in the Circle class to get the area and perimeter of the different Circle objects. Here is an example :

```
c1 = Circle(4)
c2 = Circle(6)
c3 = Circle(7)
print(c1.get_perimeter())
print(c1.get_area())
print(c2.get_perimeter())
print(c2.get_area())
print(c3.get_perimeter())
print(c3.get_area())
```