Introduction to programming using Python

Session 5

Matthieu Choplin

http://pythonguy.com/

Objectives of Session 5

- Scopes of variables
- Objects and Methods
- Class creation and object instantiation

Remainder of Session 4: Quiz (1)

- What is the keyword that we need to use if we want to use a function defined in an other file than our current file?

Remainder of Session 4: Quiz (2)

- What is a module in python?

Scope of Variables

Scope: the part of the program where the variable can be referenced.

A variable created inside a function is referred to as a **local variable**. Local variables can only be accessed inside a function. The scope of a local variable starts from its creation and continues to the end of the function that contains the variable.

In Python, you can also use **global variables**. They are created outside all functions and are accessible to all functions in their scope.

Example 1

We say that the variables defined inside a function are **local**, they are only accessible within the function

Variable defined outside the scope of a function are **global**. They are accessible inside our outside the function

```python
globalVar = 1
def f1():
    localVar = 2
    print(globalVar)
    print(localVar)
f1()
print(globalVar)
print(localVar)  # Out of scope. This gives an error
```

Example 2

A variable can be global and local in a flow of a program, like x here

As soon as a function has finished being executed, the local variables (inside the function) are destroyed ("garbage collected"). But the global variable are still accessible.

```
x = 1
def f1():
    x = 2
    print(x) # Displays 2
f1()
print(x) # Displays 1
```

Example 3

If you are defining a variable according to a condition, watch out! It is better to give a default value.

```
x = int(input("Enter a number: "))
if (x > 0):
    y = 4
print(y) # This gives an error if y is not created
```

Example 4

Are you using the variables you are defining?

```
sum = 0
for i in range(0, 5):
    sum += i
print(i)
```

Example 5

You can access the value of a global variable within the local scope but to modify it, you need to use the keyword **global**

```
x = 1
def increase():
    global x
    x =  x + 1
    print(x) # Displays 2
increase()
print(x) # Displays 2
```

Objects: illustration with the String object

- In Python, all data —including numbers and strings— are actually objects.
- An object is an entity. Each object has an **id** and a **type**.

```
>>> n = 3  # n is an integer
>>> id(n)
10914432
>>> type(n)
<class 'int'>
>>> f = 3.0  # f is a float
>>> id(f)
139757347082840
>>> type(f)
<class 'float'>
>>> s = "Welcome"  # s is a string
>>> id(s)
139757323539824
>>> type(s)
<class 'str'>
```

Methods

- You can perform operations on an object. The operations are defined using functions. The functions for the objects are called *methods* in Python. **Methods can only be invoked from a specific object**, using the dot notation

```
>>> s = "Welcome"
>>> s1 = s.lower()
>>> s1
'welcome'
>>> s2 = s.upper()
>>> s2
'WELCOME'
>>>
```

Seeing what methods are available

- You can use the **dir()** introspection function to see what methods have been defined for an object

```
>>> dir("example of string")
['__add__', '__class__', '__contains__', '__delattr__', '__dir__',
'__doc__', '__eq__', '__format__', '__ge__', '__getattribute__',
'__getitem__', '__getnewargs__', '__gt__', '__hash__', '__init__',
'__iter__', '__le__', '__len__', '__lt__', '__mod__', '__mul__',
'__ne__', '__new__', '__reduce__', '__reduce_ex__', '__repr__',
'__rmod__', '__rmul__', '__setattr__', '__sizeof__', '__str__',
'__subclasshook__', 'capitalize', 'casefold', 'center', 'count',
'encode', 'endswith', 'expandtabs', 'find', 'format', 'format_map',
'index', 'isalnum', 'isalpha', 'isdecimal', 'isdigit', 'isidentifier',
'islower', 'isnumeric', 'isprintable', 'isspace', 'istitle', 'isupper',
'join', 'ljust', 'lower', 'lstrip', 'maketrans', 'partition', 'replace',
'rfind', 'rindex', 'rjust', 'rpartition', 'rsplit', 'rstrip', 'split',
'splitlines', 'startswith', 'strip', 'swapcase', 'title', 'translate',
'upper', 'zfill']
```

The format() method

- The **format** method helps to format the string
    - Either with positional arguments:

```
text = "During session {}, " \
"we learned to use {} ".format(5, 'OOP')
```

- Or with named arguments

```
text = "During session {session_name}, " \
"we learned to use {topic} ".format(session_name=5, topic='OOP')
```

- More information about string formating: https://pyformat.info/

Creating our own Class

Syntax:

```
class NameOfTheClass:
    # the class body
```

Example

```
class Employee:
    pass
```

Creating instances of a class

Instances are objects created from the class blueprint. When we create an object of a class, we say that we instantiate the object

Example:

```
class Employee:
    pass

employee1 = Employee()
employee2 = Employee()
print(type(employee1))
print(employee1)
```

Attributes of an object

- To add an attribute we can simply add an attribute with the dot notation and assign a value to the attribute

```python
class Employee:
    pass

employee1 = Employee()
employee1.name = "Matt"
```

The special initializer method

- Instead of adding the attribute after creating our object, we can create them directly when creating the object, in the initializer method

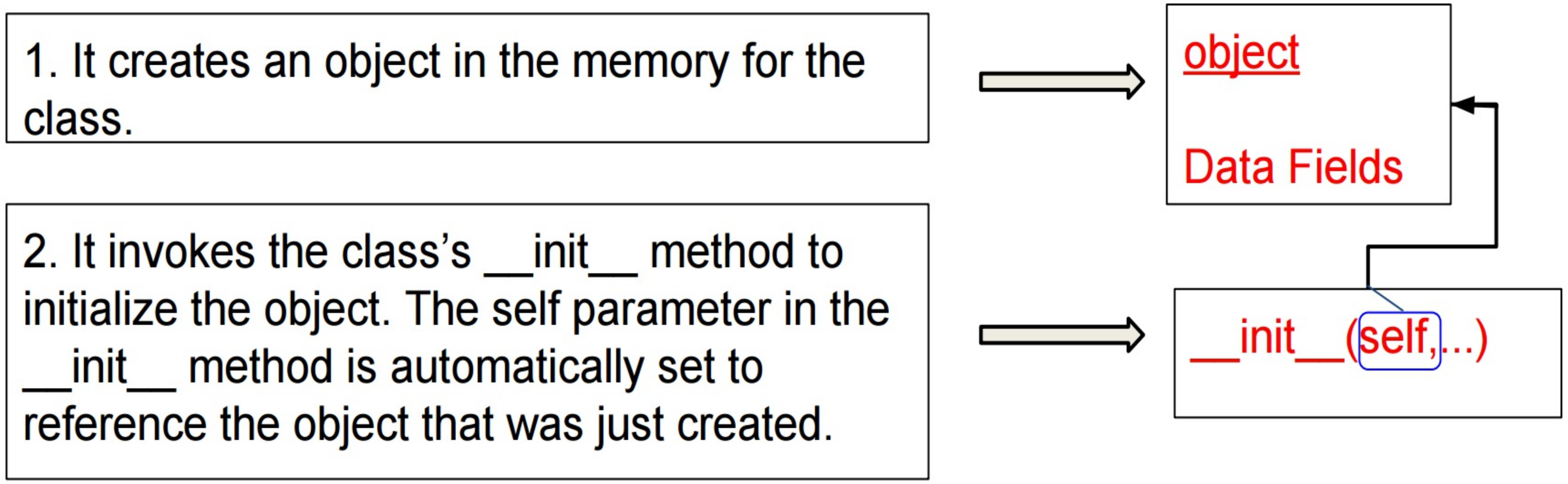
```
# class creation
class Employee:
    def __init__(self, name):
        self.name = name

#object creation
employee1 = Employee('Matt')
```

- self represents the instance/object "employee1" here

Constructing Objects

- Once a class is defined, you can create objects from the class by using the following syntax, called a **constructor**:

```
my_new_object = ClassName(optional_arguments)
```



1. It creates an object in the memory for the class.

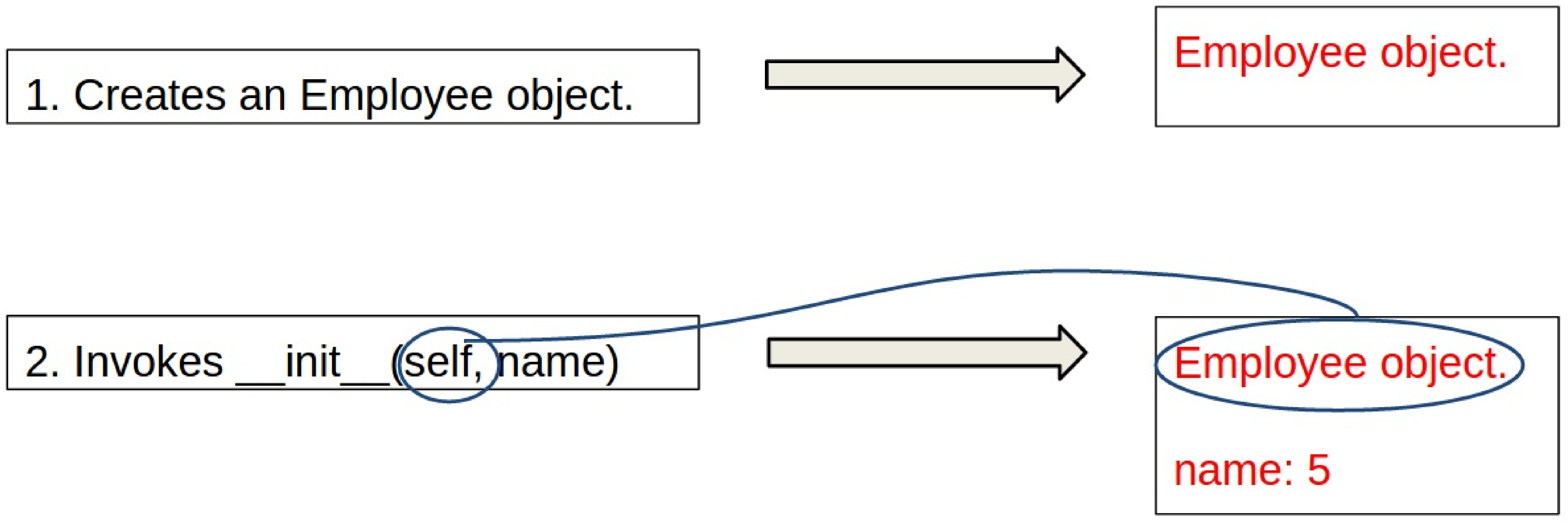
2. It invokes the class's __init__ method to initialize the object. The self parameter in the __init__ method is automatically set to reference the object that was just created.

Constructing Objects

The effect of constructing an Employee object with ...

```
employee1 = Employee(5)
```

... is shown below:


1. Creates an Employee object.
Employee object.
2. Invokes __init__(self, name)
Employee object.
name: 5

Exercise: adding attribute

- Complete the previous code to create a new attribute "date_of_birth" to the Employee class, that will be "initialized"
- Create 2 employee objects with a name and a date_of_birth and retrieve their date of birth

Class attribute

- Like Instances, Classes can also have attributes

```
class Employee:
    count = 0
    def __init__(self):
        Employee.count += 1

for i in range(3):
    Employee()

Employee.count
```

Instance Methods

- Methods are **functions defined inside a class.** They are **invoked by objects** to perform actions on the objects.
- All the methods, including the constructor have the first parameter **self**, which refers to the object that invokes the method.

```
class Employee:
    def __init__(self, name):
        self.name = name
    def generate_email_address(self):
        return self.name + "@company.com"

employee1 = Employee('Matt')
# we call the method on the object
employee1.generate_email_address()
```

Exercise: adding a method

- Complete the previous code by adding an additional method (set_name) that will set a new name to the employee
- You should call the method like so:

```
employee1.set_name("Bob")
```

- And when you retrieve the name of the employee1, it should say "Bob"

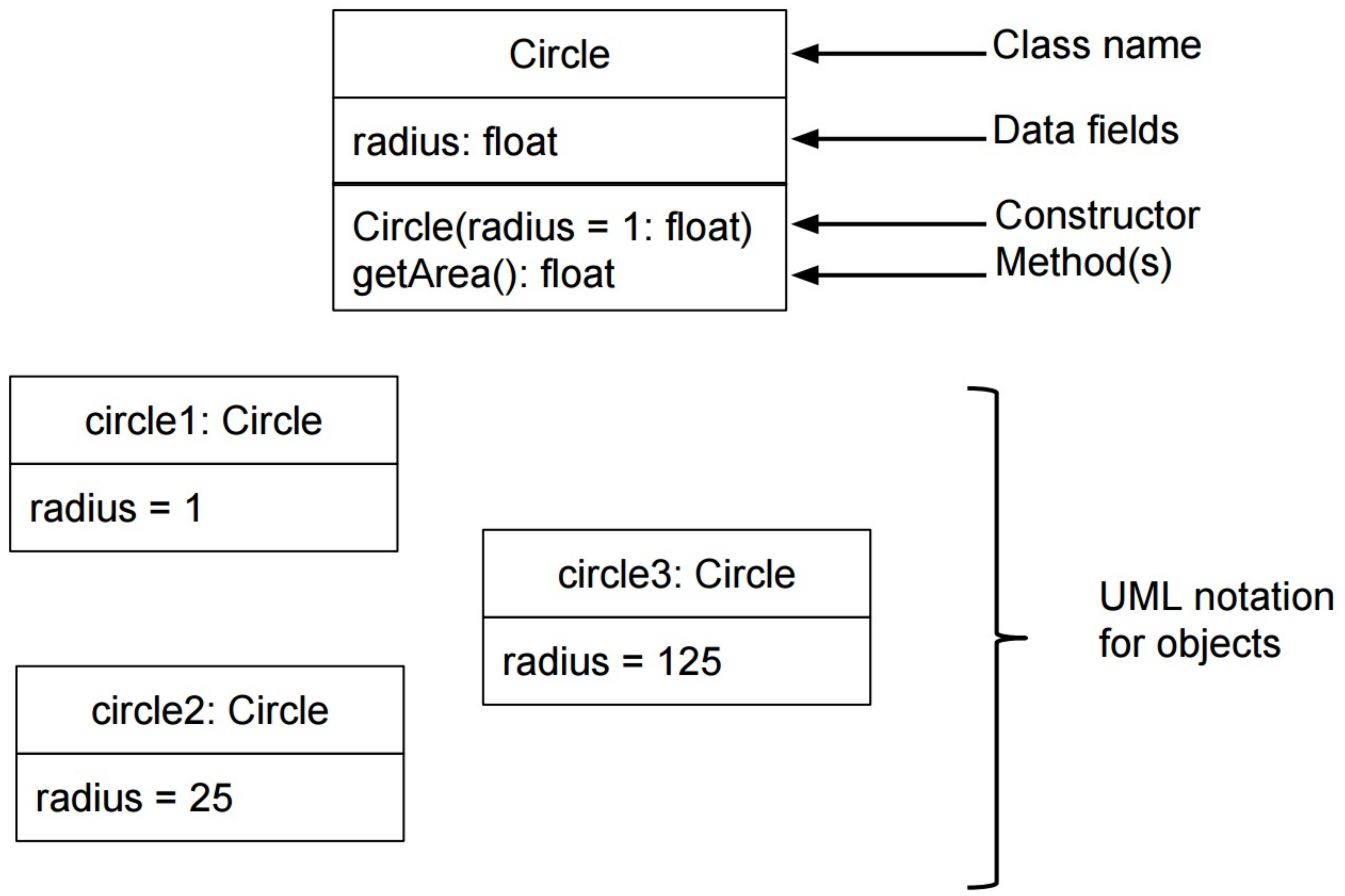
```
employee1.name
```

Accessing Objects

- After an object is created, you can access its data fields and invoke its methods using the dot operator (.)

```
# we create an employee object
employee1 = Employee('Matt')
# we call the method of the object using the dot notation
employee1.generate_email_address()
# we access one of the attribute of the object
employee.name
```

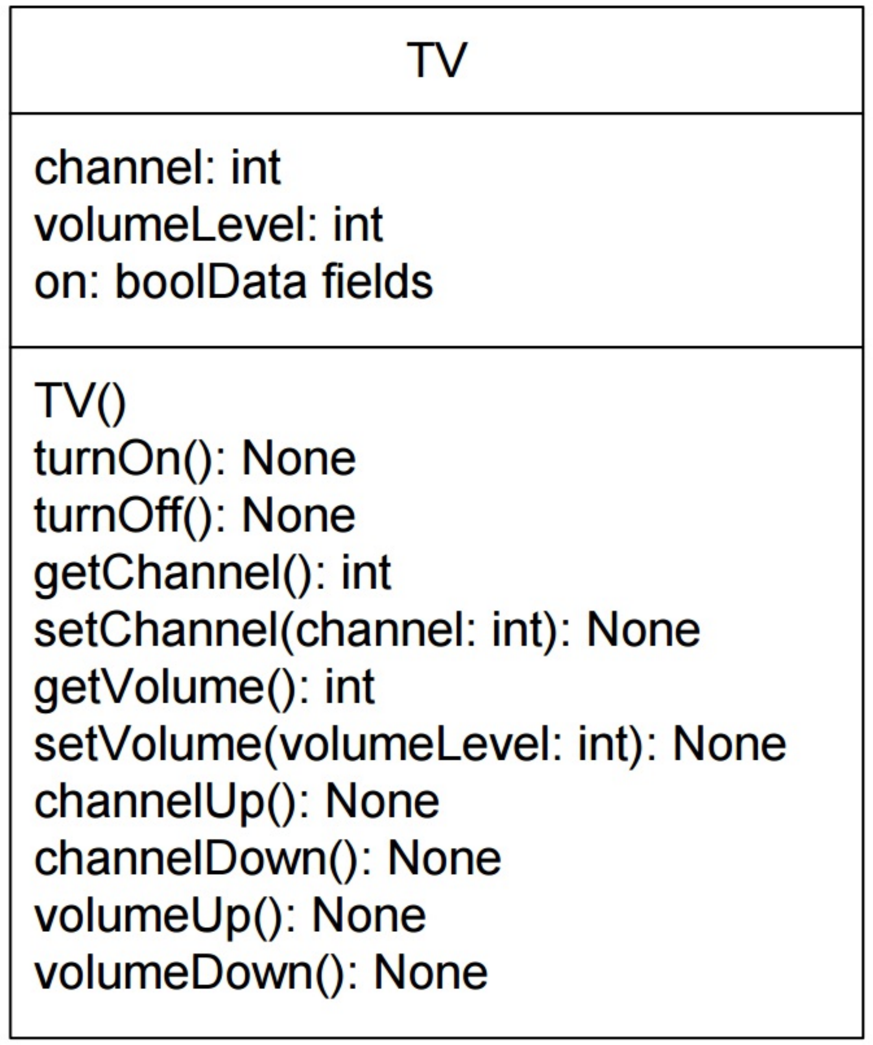
UML Class Diagram



Trace execution

See what is happening in memory with Pythontutor (click link)

Example: Defining Classes and Creating Objects



TV	
channel: int	The current channel (1 to 120) of this TV.
volumeLevel: int	The current volume level (1 to 7) of this TV.
on: boolData fields	Indicates whether this TV is on/off.
TV()	Constructs a default TV object.
turnOn(): None	Turns on this TV.
turnOff(): None	Turns off this TV.
getChannel(): int	Returns the channel for this TV.
setChannel(channel: int): None	Sets a new channel for this TV.
getVolume(): int	Gets the volume level for this TV.
setVolume(volumeLevel: int): None	Sets a new volume level for this TV.
channelUp(): None	Increases the channel number by 1.
channelDown(): None	Decreases the channel number by 1.
volumeUp(): None	Increases the volume level by 1.
volumeDown(): None	Decreases the volume level by 1.

Example: Defining Classes and Creating Objects

- See TV.py
- See TestTV.py

Exercise - The Rectangle class

Following the example of the Circle class, design a class named Rectangle to represent a rectangle. The class contains:

- Two data fields named width and height.
- A constructor that creates a rectangle with the specified width and height. The default values are 1 and 2 for the width and height, respectively.
- A method named getArea() that returns the area of this rectangle
- A method named getPerimeter() that returns the perimeter

Implement the class. Write a test program that creates two Rectangle objects—one with width 4 and height 40 and the other with width 3.5 and height 35.7. Display the width, height, area, and perimeter of each rectangle in this order.

Exercise: The Account class

Design a class named Account that contains:

- A field named id for the account.
- A field named balance for the account.
- A field named annualInterestRate that stores the current interest rate.
- A constructor that creates an account with the specified id (default 0), initial balance (default 100), and annual interest rate (default 0).
- A method named getMonthlyInterestRate() that returns the monthly interest rate.
- A method named getMonthlyInterest() that returns the monthly interest.
- A method named withdraw that withdraws a specified amount from the account.
- A method named deposit that deposits a specified amount to the account.
- Create a method getBalance that will show the message: "The current balance of the account 0 is 100" (use format).