CLASSES

CLASSES

- If you have done anything in computer science before, you likely will have heard the term object oriented programming (OOP) (hlutbundin forritun)
- If you haven´t then we should answer the question What is OOP?
  - The short answer is that object oriented programming is a way to think about "objects" in a program (such as variables, functions, etc)
  - A program becomes less a list of instruction and more a set of objects and how they interact with each other

CLASSES

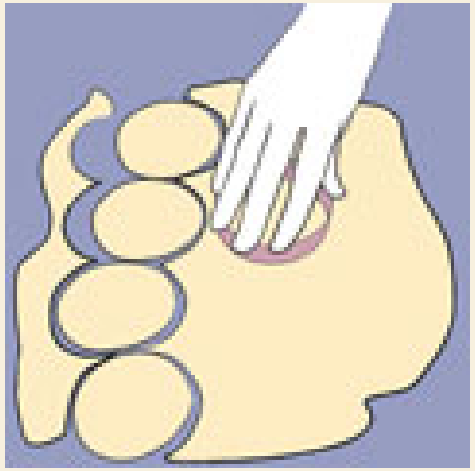
- There are 4 main principles of OOP
  - ***encapsulation***: hiding design details to make the program clearer and more easily modified later
  - ***modularity***: the ability to make objects stand alone so they can be reused (our modules, like the math module)
  - ***inheritance***: create a new object by inheriting (like father to son) many object characteristics while creating or over-riding for this object
  - ***polymorphism***: (hard) Allow one message to be sent to any object and have it respond appropriately based on the type of object it is.

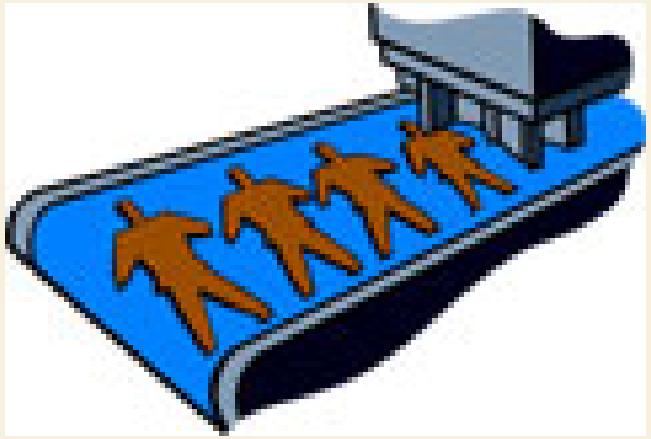
CLASSES

- Everything in Python is an object!
  - Thus Python embraces OOP at a fundamental level
- We have seen that lists, dictionaries, sets, etc. are objects
- But we can make our own object
  - To do that we use Classes!

CLASSES

- What is a class?
  - A class blueprint of a data structure which can hold data and methods that work on that data
  - When the blueprint is ready we can create instances of that class!





CLASSES

- What is an instance?
  - Consider the following code snippets

This code snippet has 3 list instances

```
1    a_list = [1, 2, 3]
2    b_list = [7, 8, 9]
3    c_list = [12, 4, 3]
```

This code snippet has 3 integer instances

```
1    a = 3
2    b = 4
3    c = 6
```

CLASSES

- One of the harder things to understand is what a class is and what an instance of a class is
- The analogy of the cookie cutter and a cookie is a famous one
  - Before you can make cookies you must have a well-defined cookie cutter
  - In this analogy the cookie cutter is the class (a blueprint of a cookie) and the cookies are instances of the class

CLASSES

- Why use classes?
  - We make classes because we often need more complicated, user-defined data types to construct data models we can use
  - Classes are very useful when we need to create data models of real world things such as cars, vehicles, etc.
  - Each class has potentially two aspects:
    - The data that each instance might contain
    - Methods (functions)

CLASSES

- The standard way to name a class in Python is called ***CamelCase***:
  - Each word of a class begins with a Capital letter
  - No underlines
  - This makes recognizing a class easier

CLASSES

- The constructor method
  - When a class is defined, we usually create a special methods called __init__
    - The double underscore is sometimes called dunder
  - This method is called the *constructor*. It is this method that creates an instance of the class
  - By assigning values in the constructor method, every instance will start out with the same variables

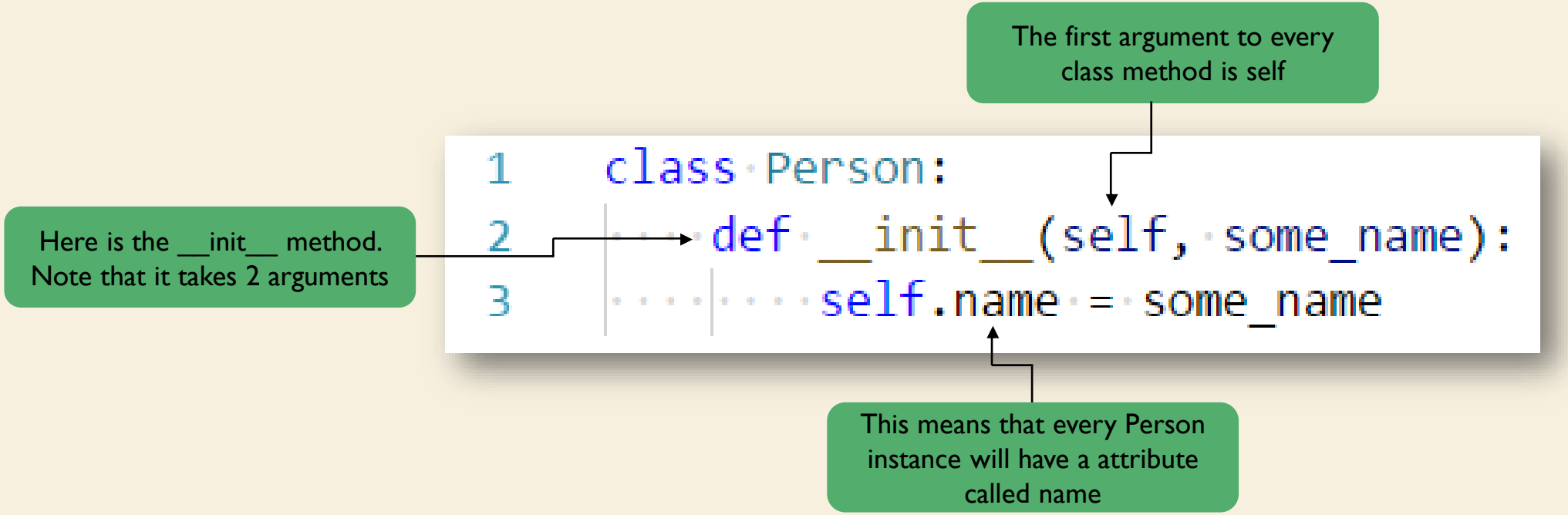
- Do note that you never call the __init__ method, Python takes care of that for us internally

CLASSES

- Do note that if you don't provide a constructor method, then a default constructor is provided
- The default constructor does various system tasks to create a class instance, nothing more
- You cannot pass arguments to the default constructor

CLASSES

- Here is an example of a class called Person



CLASSES

- Classes can have data and methods
  - The data is called attributes
  - A class can have many attributes
  - These attributes are initialised and declared in the __init__ method
  - The Person class has one attribute called name
    - This means that every instance of the class Person will have a name attribute

```
1   class Person:
2       def __init__(self, some_name):
3           self.name = some_name
```

CLASSES

We never pass anything to the self parameter, that is done automatically for us

```
1   class Person:
2       def __init__(self, some_name):
3           self.name = some_name
4
5
6
7   def main():
8       person_a = Person("John")
9       person_b = Person("Alice")
10
11  main()
```

When Python sees this line of code it automatically calls the __int__ method and passes it the string John. The paramenter some_name gets assigned the value John and sets the name attribute as John

This is how we create instances of the Person class

CLASSES

- We can access the class attributes by using dot notation

```
1   class Person:
2       def __init__(self, some_name):
3           self.name = some_name
4
5
6
7   def main():
8       person_a = Person("John")
9       person_b = Person("Alice")
10
11      print(person_a.name) # prints: John
12      print(person_b.name) # prints: Alice
13
14  main()
```

CLASSES

- We can define as many methods (functions) as we wish in a class
- Do note that the first parameter to all class methods must be self!
- Class methods have access to the class attributes
- We can access the class methods using dot notation

```
class Person:
    def __init__(self, some_name):
        self.name = some_name

    def print_name_uppercase(self):
        print(self.name.upper())


def main():
    person_a = Person("John")
    person_b = Person("Alice")

    print(person_a.name) # prints: John
    print(person_b.name) # prints: Alice

    person_a.print_name_uppercase() # prints JOHN
    person_b.print_name_uppercase() # ALICE

main()
```

CLASSES

- Another special method is the __str__ method
- This is a method that returns the string representation of the class
- This is the method that the print function will look for when printing a class instance

```
1   class Person:
2       def __init__(self, some_name):
3           self.name = some_name
4
5       def __str__(self):
6           return "Name: {}".format(self.name)
7
8       def print_name_uppercase(self):
9           print(self.name.upper())
10
11
12  def main():
13      person_a = Person("John")
14      person_b = Person("Alice")
15
16      print(person_a) # prints: Name: John
17      print(person_b) # prints: Name: Alice
18
19  main()
```

CLASSES

- The __str__ must accept self as the first parameter and it must **return** a string
- Like all other class methods __str__ has access to the class attributes

```python
class Person:
    def __init__(self, some_name):
        self.name = some_name

    def __str__(self):
        return "Name: {}".format(self.name)

    def print_name_uppercase(self):
        print(self.name.upper())


def main():
    person_a = Person("John")
    person_b = Person("Alice")

    print(person_a) # prints: Name: John
    print(person_b) # prints: Name: Alice

main()
```

The print function calls the __str__ method of the Person class because these variables are Person instances