

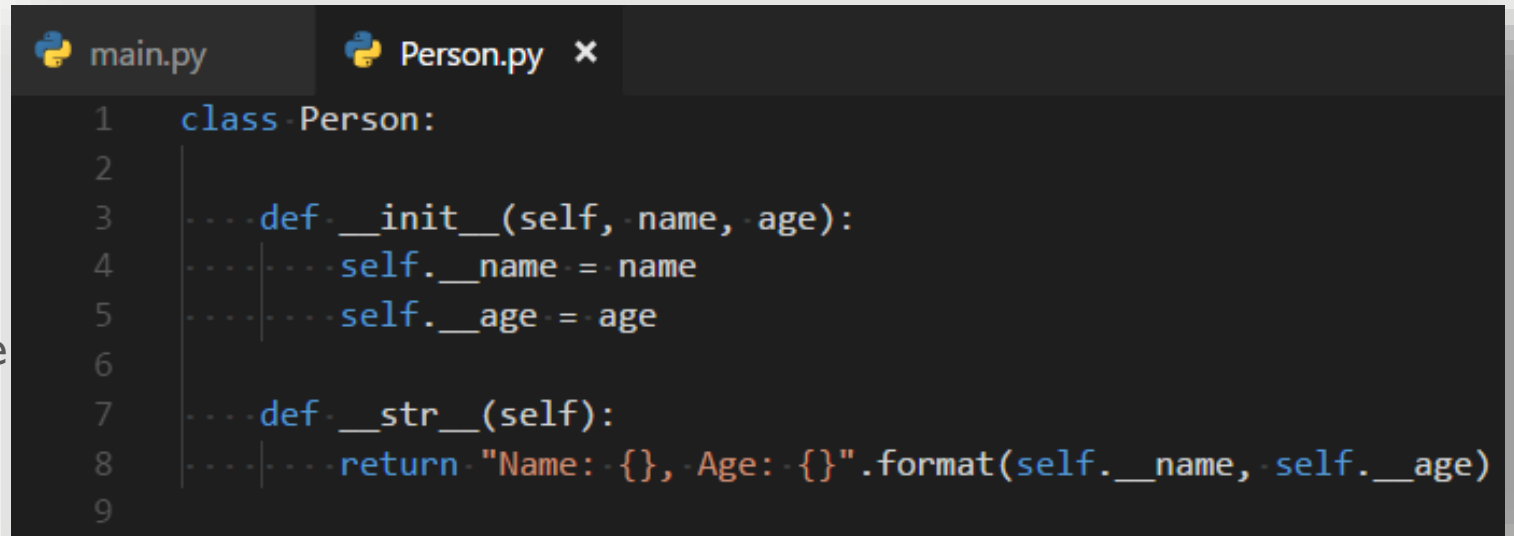
CLASSES AND FILES

CLASSES

- So far you have seen that lines of code can grow very fast
 - The bigger the system is = more lines of code
- So far we have been using a single file for all our code
 - This is usually not the way to go
- Lets start with classes
 - Every class should reside in its own file
 - Lets see an example on the next slide

CLASSES

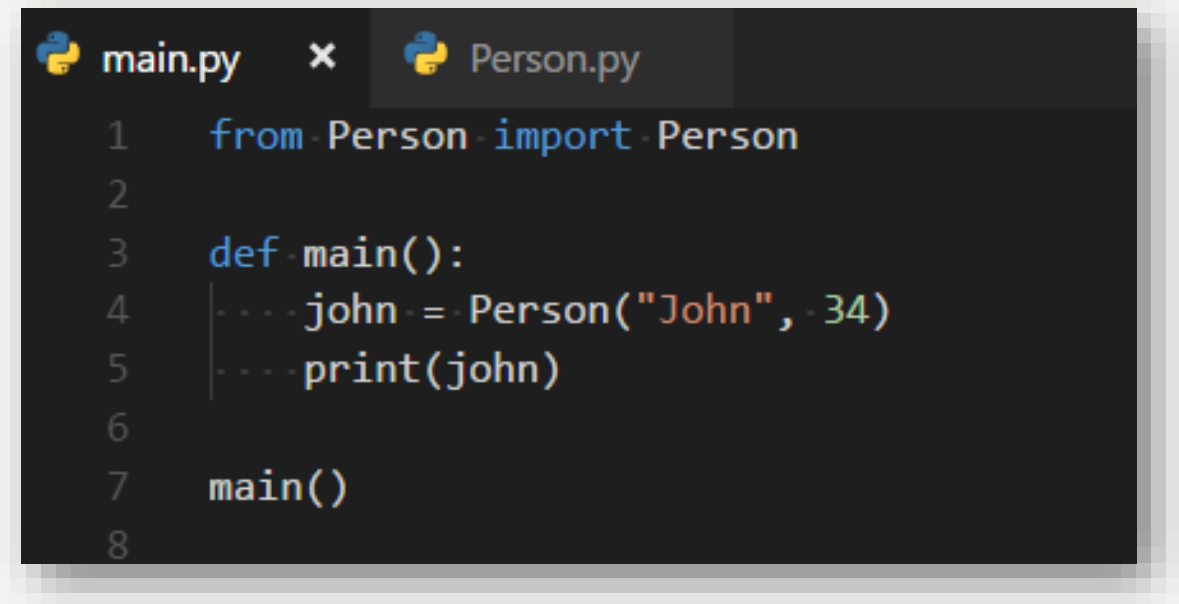
- Consider the class Person
- It is a simple class and it is contained in a file called Person.py
 - Note that it is good practice to name the file with the same name as the class it holds



```
main.py Person.py x
1 class Person:
2
3     def __init__(self, name, age):
4         self.__name = name
5         self.__age = age
6
7     def __str__(self):
8         return "Name: {}, Age: {}".format(self.__name, self.__age)
9
```

CLASSES

- We can now import the Person class to another file and use it there
- In the first line we import the Person class from the person file
- Then we can create instances like is shown in line 4
- For a tiny project this is not necessary but as a project grows it becomes very important to organise the code base in separate files

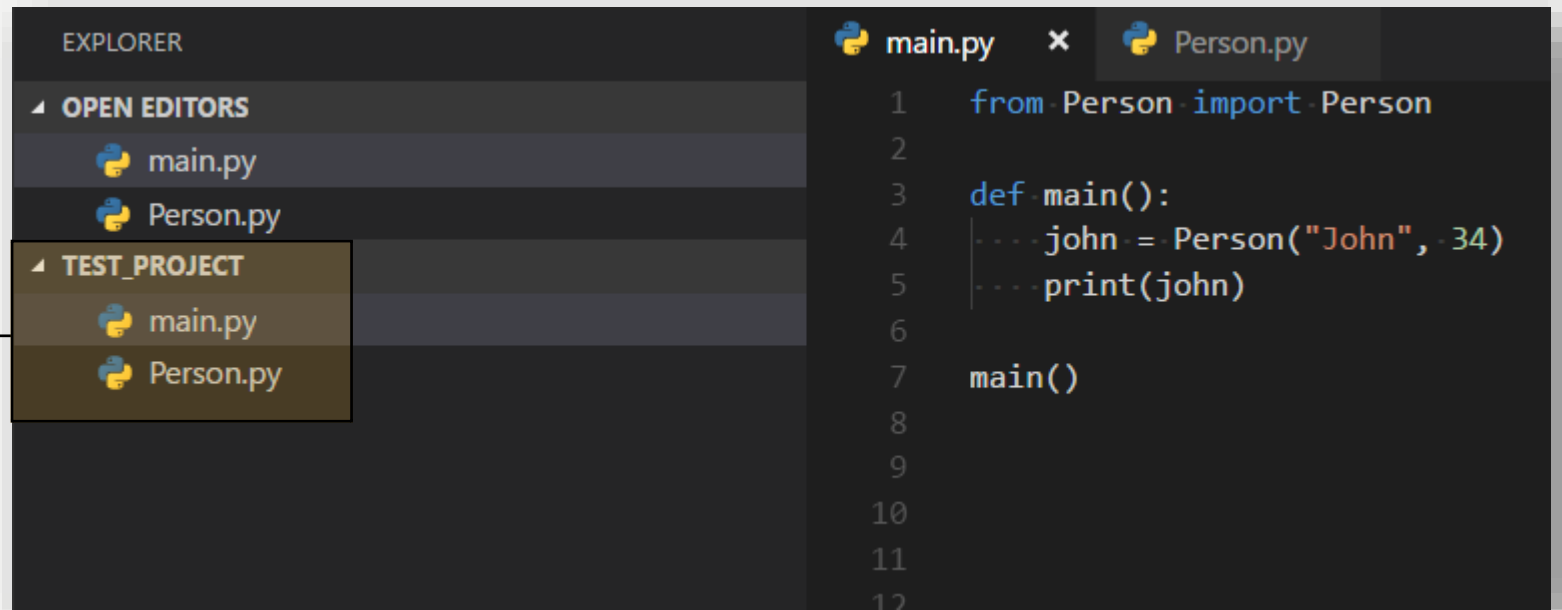


```
main.py  x  Person.py
1  from Person import Person
2
3  def main():
4      john = Person("John", 34)
5      print(john)
6
7  main()
8
```

CLASSES

- It is as easy as this to import a file to another file in python but only if the two files are in the same directory/folder

Person.py and main.py
both reside in the
directory test_project



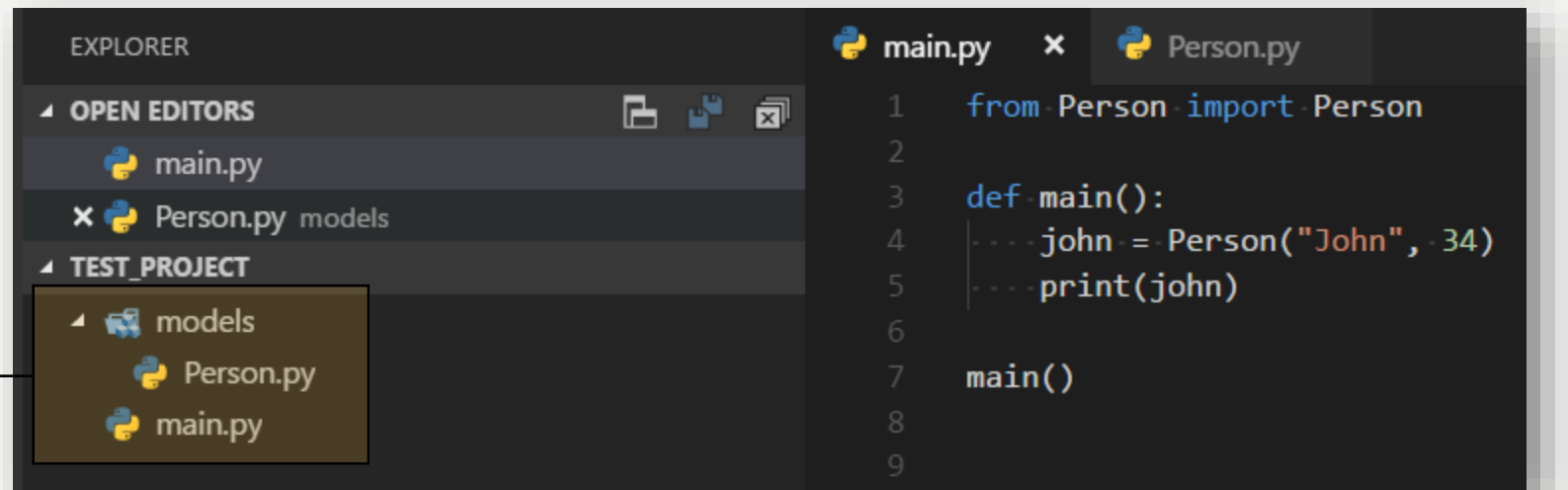
The screenshot shows the Visual Studio Code interface. On the left, the 'EXPLORER' sidebar displays a project structure with a folder named 'TEST_PROJECT' containing two files: 'main.py' and 'Person.py'. The 'OPEN EDITORS' sidebar shows both files are open. The main editor area displays the code in 'main.py', which imports 'Person' from 'Person.py' and calls the 'main()' function. The code is as follows:

```
1 from Person import Person
2
3 def main():
4     john = Person("John", 34)
5     print(john)
6
7 main()
8
9
10
11
12
```

CLASSES

- However, it is quite common to divide the code base into subfolders
 - For example, let's keep all our classes that represent some entity in a folder called models
 - It is common to call classes that represent some entity a model class (more on that later)

Here we have created a subfolder called models. Within it resides Person.py

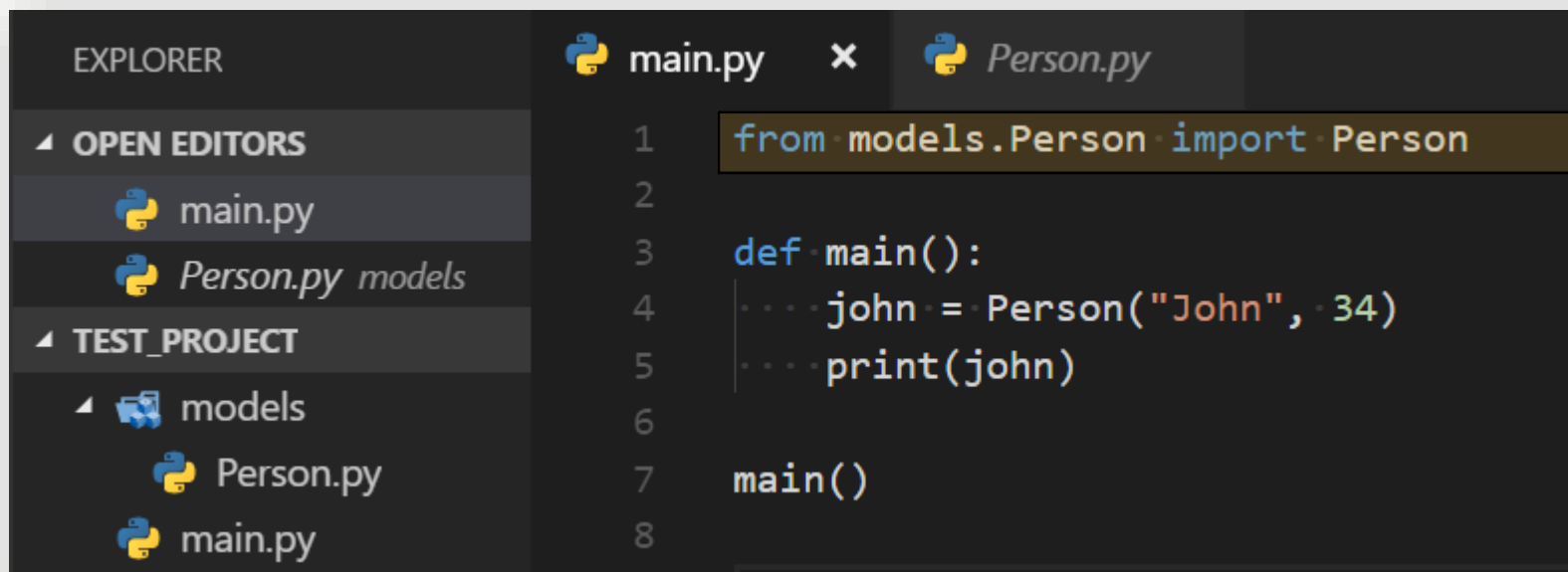


The screenshot shows a code editor interface. On the left, the 'EXPLORER' sidebar displays the project structure. Under 'TEST_PROJECT', there is a subfolder named 'models' which contains two files: 'Person.py' and 'main.py'. The 'main.py' file is currently open in the editor, showing the following code:

```
1 from Person import Person
2
3 def main():
4     john = Person("John", 34)
5     print(john)
6
7 main()
8
9
```

CLASSES

- The next step is to change the import statement at the top of main.py



The screenshot shows a code editor interface. On the left, the 'EXPLORER' panel shows a project structure with a folder named 'models' containing 'Person.py' and 'main.py'. The 'OPEN EDITORS' panel shows 'main.py' and 'Person.py models'. The main editor window shows the code in 'main.py' with the following content:

```
1 from models.Person import Person
2
3 def main():
4     john = Person("John", 34)
5     print(john)
6
7 main()
8
```

The first line, `from models.Person import Person`, is highlighted with a yellow background.

Now the path contains the name of the folder followed by the name of the file we wish to import

CLASSES

- It is however worth noting that before Python version 3.3.x importing code wasn't so easy
 - You had to add an empty file called `__init__.py` to the subdirectory / subfolder
 - It is quite likely that you will see this `__init__.py` file when viewing older code bases

Here we've added a file called `__init__.py` and as you can see it is completely empty

