INFO5002: Intro to Python for Info Sys

Week 7

Slides created by: Zachary Doucet

LVX VERITAS VIRTVS

Northeastern University

Week 7

I. Testing

II. I/O

III. Scraping

Recap

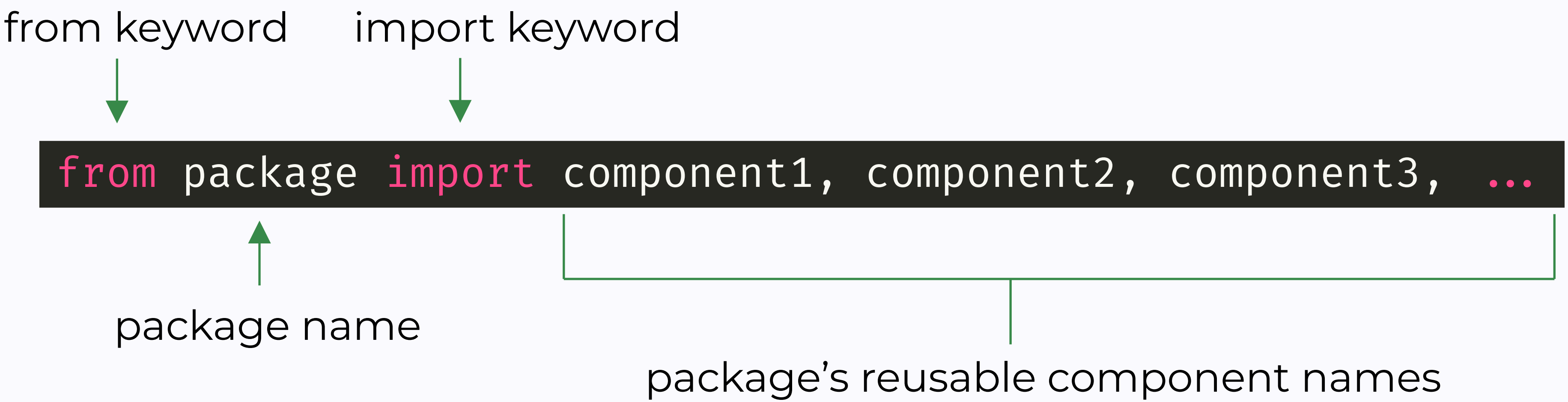
Encapsulation

- Expose minimally.

```
class BankAccount:
    def __init__(self, initial):
        self.__balance = initial

    def deposit(self, amt):
        self.__balance += amt

    def withdraw(self, amt):
        if amt < self.__balance:
            self.__balance -= amt
```

- Public (everyone)
- Protected (me and my children)
- Private (only me)

We import with from import



from keyword

import keyword

```
from package import component1, component2, component3, ...
```

package name

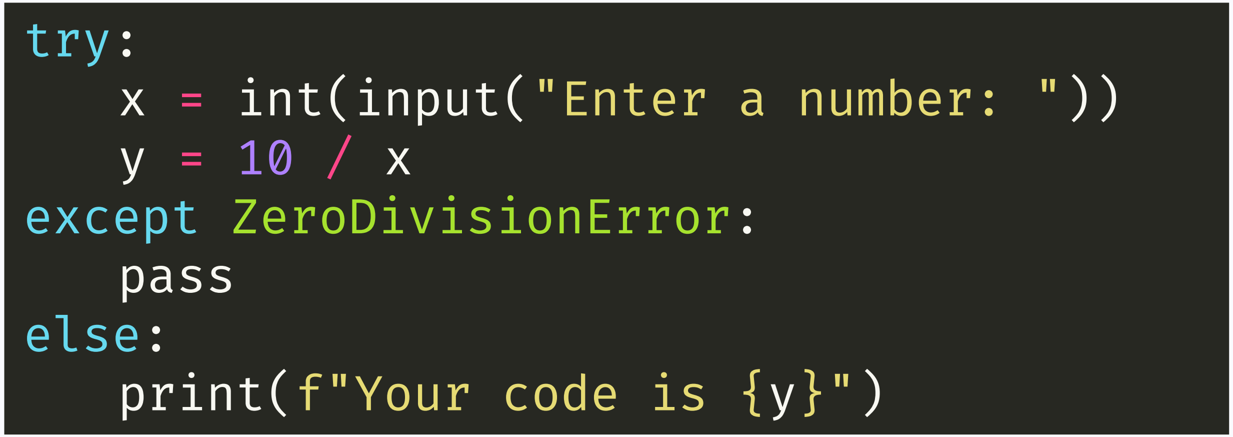
package’s reusable component names

Module vs package vs sub-package.

```
from package import *
```

We protect our at risk with try-except.

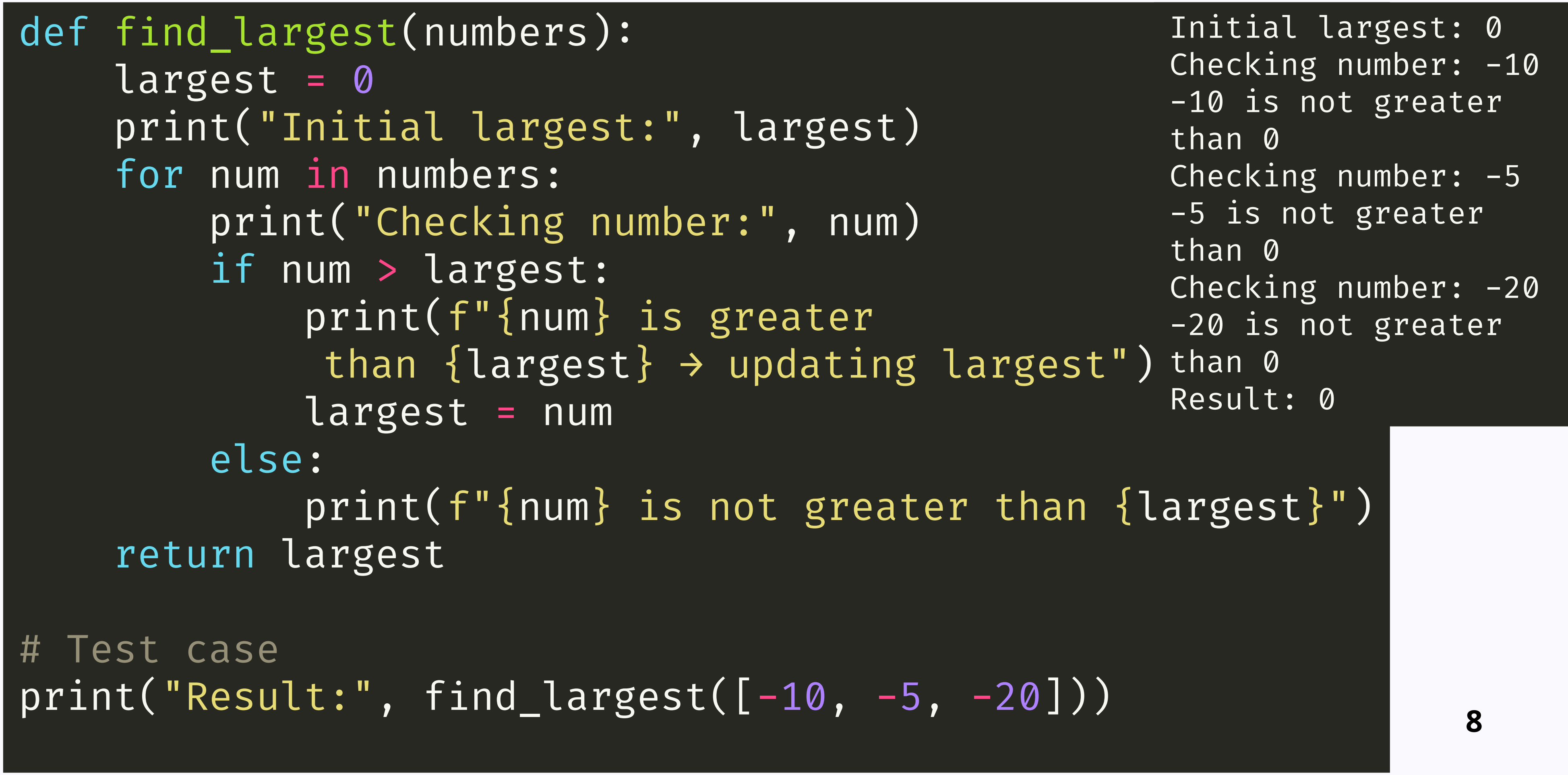
- Code that can produce exceptions should be protected.



```python
try:
    x = int(input("Enter a number: "))
    y = 10 / x
except ZeroDivisionError:
    pass
else:
    print(f"Your code is {y}")
```

Debugging

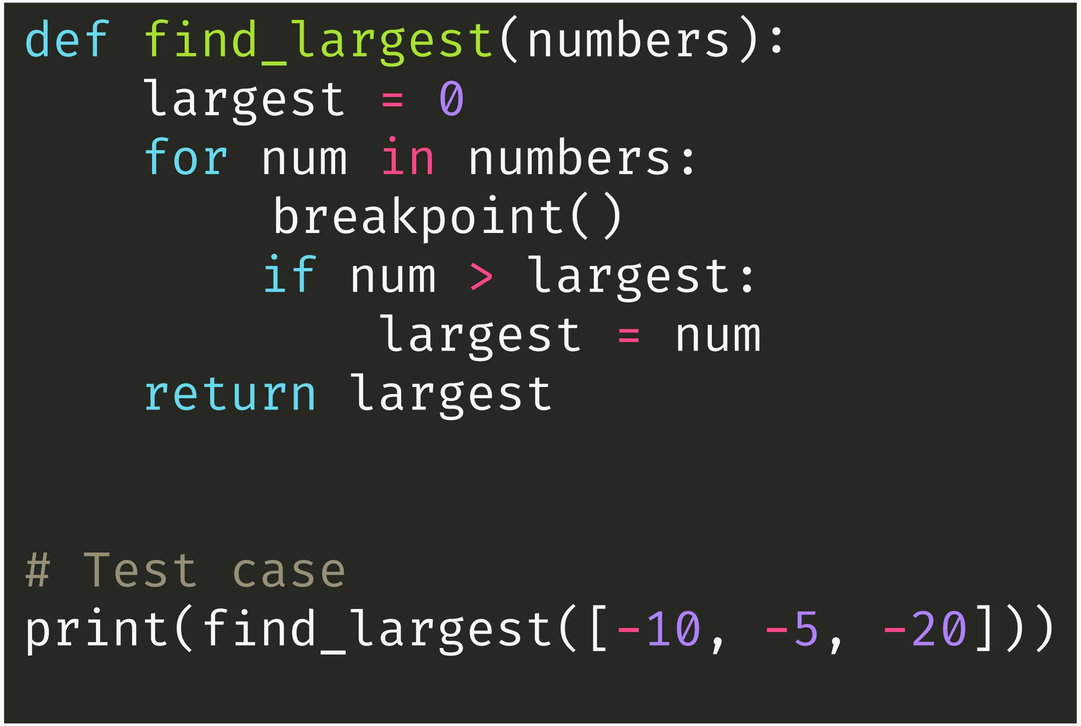
Adding print statements



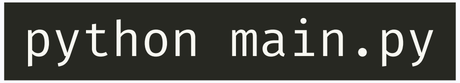
```python
def find_largest(numbers):
    largest = 0
    print("Initial largest:", largest)
    for num in numbers:
        print("Checking number:", num)
        if num > largest:
            print(f"{num} is greater than {largest} → updating largest")
            largest = num
        else:
            print(f"{num} is not greater than {largest}")
    return largest

# Test case
print("Result:", find_largest([-10, -5, -20]))
```

```
Initial largest: 0
Checking number: -10
-10 is not greater
than 0
Checking number: -5
-5 is not greater
than 0
Checking number: -20
-20 is not greater
than 0
Result: 0
```

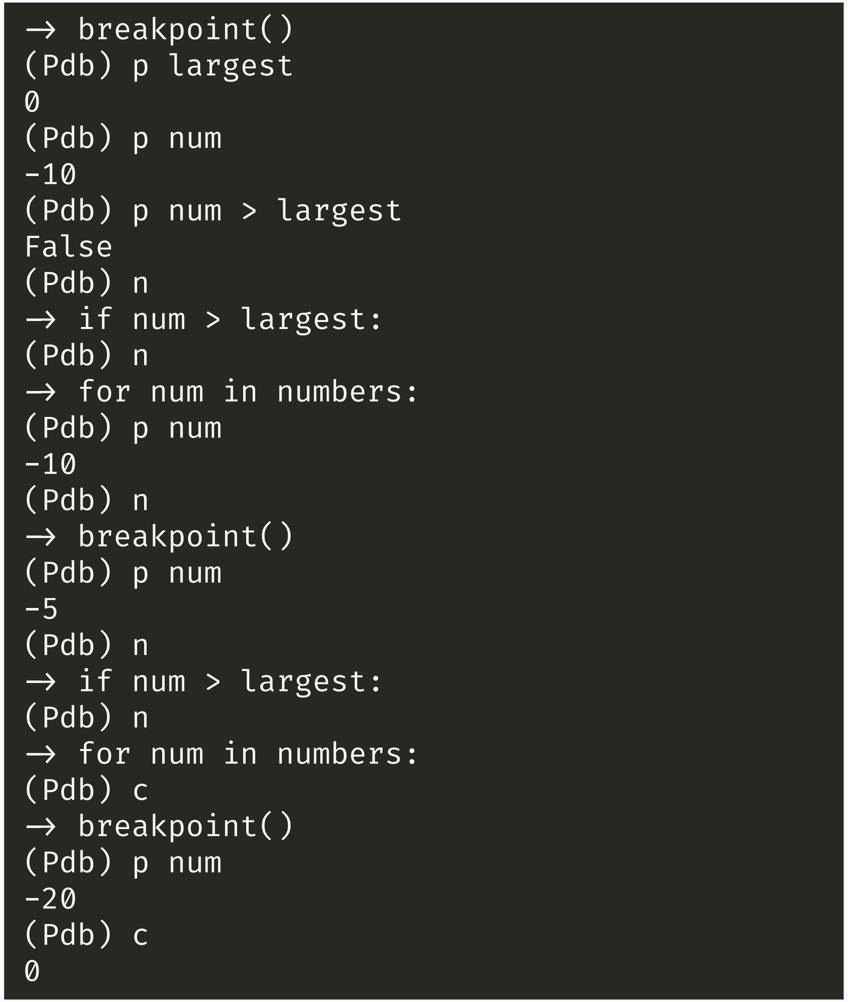
Using Python pdb



```python
def find_largest(numbers):
    largest = 0
    for num in numbers:
        breakpoint()
        if num > largest:
            largest = num
    return largest


# Test case
print(find_largest([-10, -5, -20]))
```



```
python main.py
```



```
-> breakpoint()
(Pdb) p largest
0
(Pdb) p num
-10
(Pdb) p num > largest
False
(Pdb) n
-> if num > largest:
(Pdb) n
-> for num in numbers:
(Pdb) p num
-10
(Pdb) n
-> breakpoint()
(Pdb) p num
-5
(Pdb) n
-> if num > largest:
(Pdb) n
-> for num in numbers:
(Pdb) c
-> breakpoint()
(Pdb) p num
-20
(Pdb) c
0
```

Testing

PCC 209-223

Unit tests

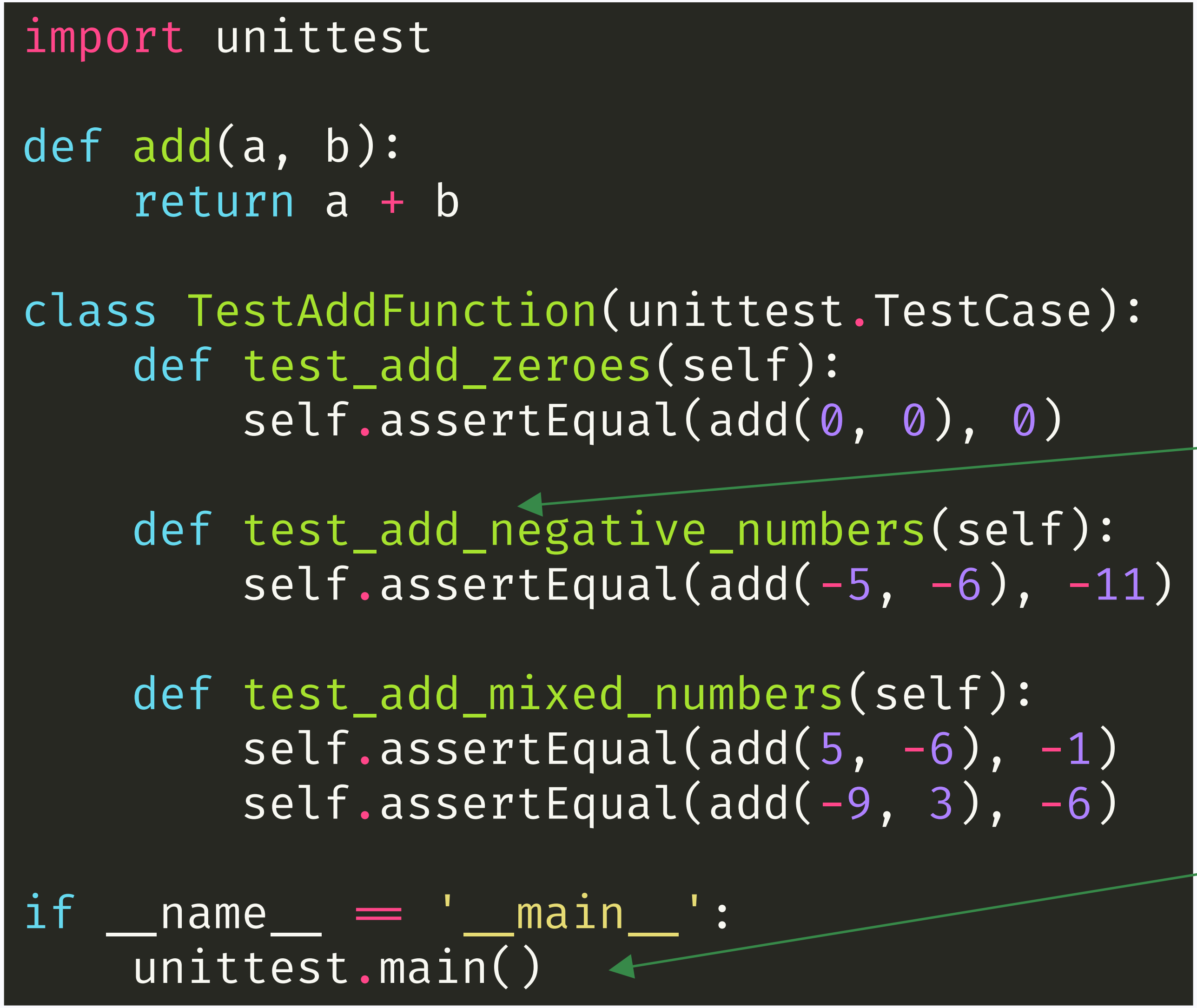
- Test the smallest functional unit.
- Tests should be independent.
- Sum of tests should cover as many lines of code (have high coverage).

Python unittest

- assertEqual(a, b): see if a == b
- assertTrue(x): see if bool(x) == True
- assertIsInstance(a, b): see if a is instance of b
- assertIsNone(x): see if x == None
- assertFalse(x): see if bool(x) == False
- assertIs(a, b): see if a is b
- assertIn(a, b): see if a in b

Concepts

- Test case: individual unit of testing.
- Test suite: group of test cases and test suites.
- Test fixture: any code that runs before or after tests to prepare or cleanup.
- Test runner: orchestrates the execution of tests and returns result to user.

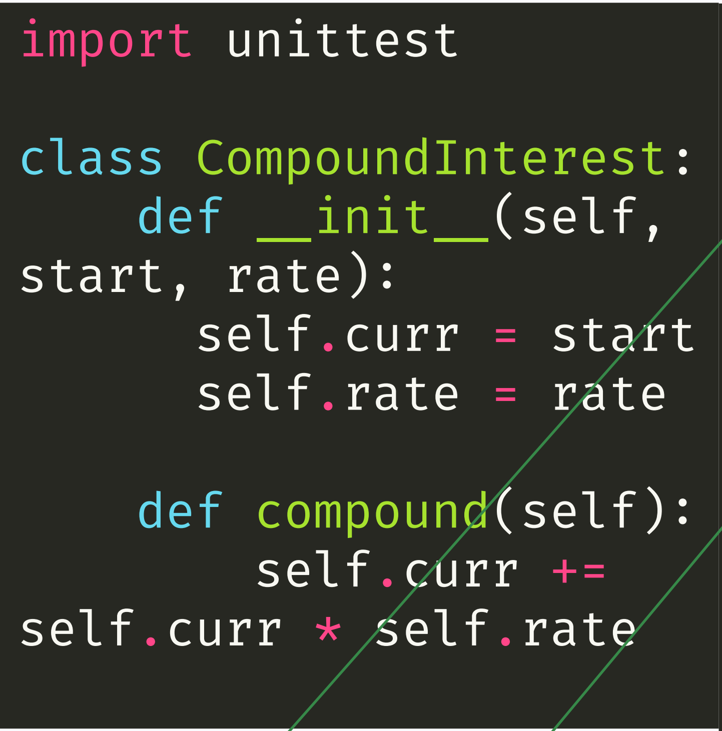


```python
import unittest

def add(a, b):
    return a + b

class TestAddFunction(unittest.TestCase):
    def test_add_zeroes(self):
        self.assertEqual(add(0, 0), 0)

    def test_add_negative_numbers(self):
        self.assertEqual(add(-5, -6), -11)

    def test_add_mixed_numbers(self):
        self.assertEqual(add(5, -6), -1)
        self.assertEqual(add(-9, 3), -6)

if __name__ == '__main__':
    unittest.main()
```

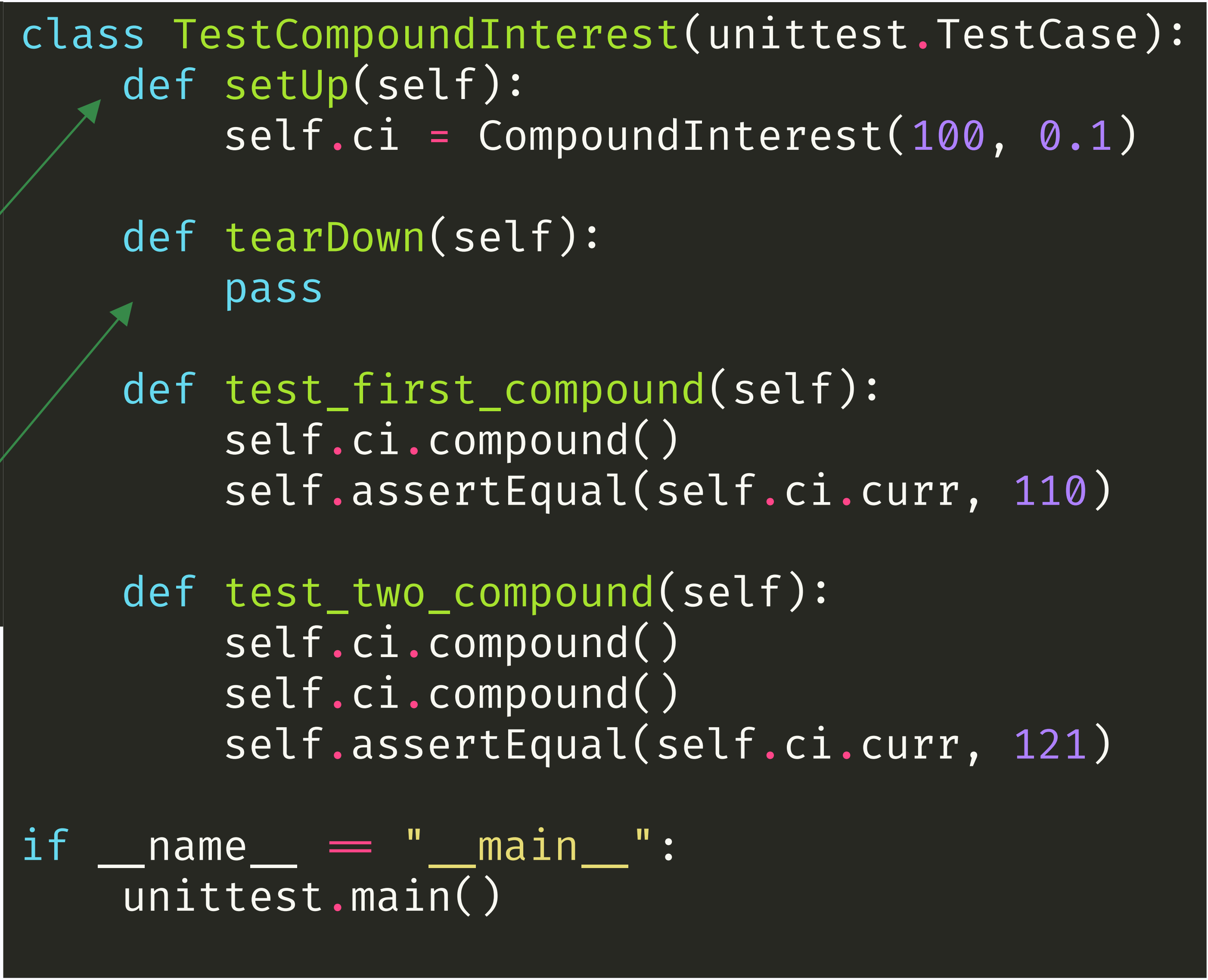
A test case named TestAddFunction with 3 unit tests.

Individual unit test must start with the letters test_ and have at least 1 assert.

Will run all test cases that inherit unittest.TestCase



```
import unittest

class CompoundInterest:
    def __init__(self,
start, rate):
        self.curr = start
        self.rate = rate

    def compound(self):
        self.curr +=
self.curr * self.rate
```

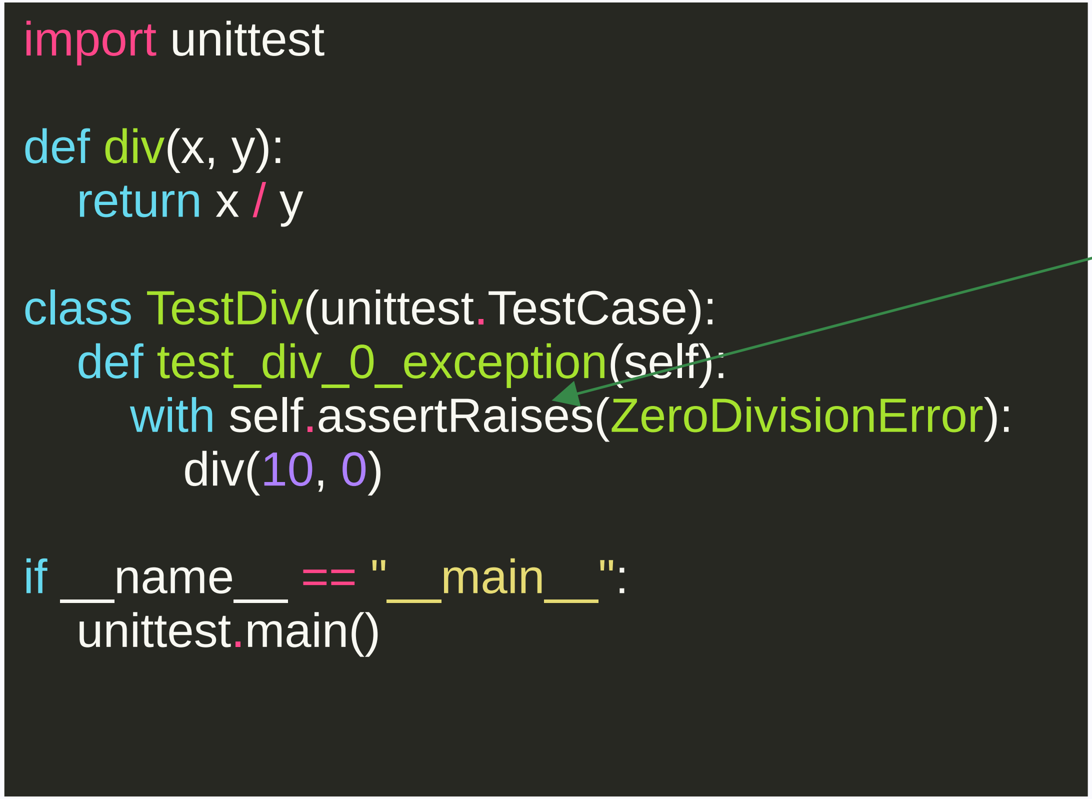


```
class TestCompoundInterest(unittest.TestCase):
    def setUp(self):
        self.ci = CompoundInterest(100, 0.1)

    def tearDown(self):
        pass

    def test_first_compound(self):
        self.ci.compound()
        self.assertEqual(self.ci.curr, 110)

    def test_two_compound(self):
        self.ci.compound()
        self.ci.compound()
        self.assertEqual(self.ci.curr, 121)

if __name__ == "__main__":
    unittest.main()
```

Code runs before each test

Code runs after each test

Testing for exceptions



```
import unittest

def div(x, y):
    return x / y

class TestDiv(unittest.TestCase):
    def test_div_0_exception(self):
        with self.assertRaises(ZeroDivisionError):
            div(10, 0)

if __name__ == "__main__":
    unittest.main()
```

Create a block of **self.assertRaises(ExceptionType)** to test if the block throws the exception

Let’s practice

In canvas you will find a file with some code written. I want you to create a new file **test.py** which will have unit tests that together have 100% coverage of the file’s functionalities.

Why bother?

- If we test out code as we go why create unit tests?
- Because if we change it later we may break it.
    - Unit tests allow us to detect if new code breaks existing behaviour.

I/O

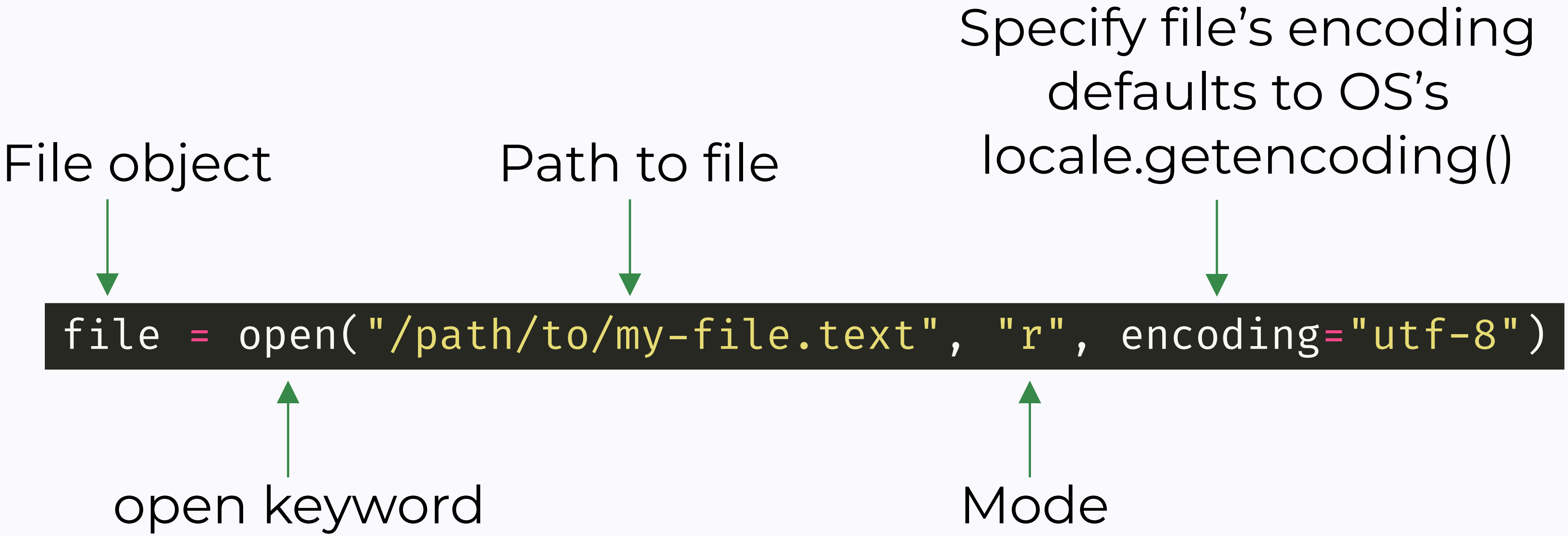
So far all data has been in memory

- We have been creating variables that hold data in our code.
- This data is not persistent and when the program stops running all of the data is released.

Keep persistence with IO

- Input/Output (I/O) acts as an **interface** between your code and the operating system's file system.
- Three types:
  - text (string)
  - binary (bytes): can store non-text data
  - Raw: rarely used

Load text with open r



Specify file’s encoding defaults to OS’s locale.getencoding()

File object

Path to file

```
file = open("/path/to/my-file.text", "r", encoding="utf-8")
```

open keyword

Mode

File modes

Character	**Meaning**
r	reading (default)
w	writing, deleting old first
x	create new file, fail if already existing
a	writing, appending to old
b	binary mode
t	text mode (default)
+	suffix, to add on read or write r+ (read and write w/out deleting file first) w+ (read and write deleting file first)

Encodings

- Important to specify the file’s encoding that you are loading. Otherwise your data will be unrecognisable at worst or crash at best.

- Popular encodings: ascii, utf-8, utf-16, utf-32.

Encoding matters example

```python
file = open("test_file.txt", "w", encoding="utf-8")
file.write("Welcome to class")
file.close()

file2 = open("test_file.txt", "r", encoding="utf-8")
print(file2.read())
file2.close()

file3 = open("test_file.txt", "r", encoding="utf-16")
print(file3.read())
file3.close()
```

```
  File "/usr/lib64/python3.13/encodings/utf_16.py", line 67, in _buffer_decode
    raise UnicodeDecodeError("utf-16", input, 0, 2, "Stream does not start with BOM")
UnicodeDecodeError: 'utf-16' codec can't decode bytes in position 0-1: Stream does not start with BOM
```

Load binary with open rb

```
file = open("/path/to/my-file.text", "rb")
```

Load raw with open rb no buffering

```
file = open("/path/to/my-file.text", "rb", buffering=0)
```

What to do with file objects?

- close(): close the file
- readline(): read one line from file
- readlines(): return list of lines from file
  - for line in file:
- read(size=-1): read size number of bytes
- write(data): write data

Move file cursor

- seek(offset, whence=os.SEEK_SET): change the stream number of bytes from relative position set with whence.
  - os.SEEK_SET: start of stream (offset >= 0)
  - os.SEEK_CUR: current position
  - os.SEEK_END: end of stream (offset <= 0, usually)

Let’s practice

Create a function **process_student_grades** that loads a file you will find on canvas and creates a new file **output.txt** which has number of 0-9% on one line, number of 10-19% on one line, …, number of 90-99% on one line, number of 100% on one line, and the average on the final line. Use utf-8 encoding.

It is annoying have one data per line!

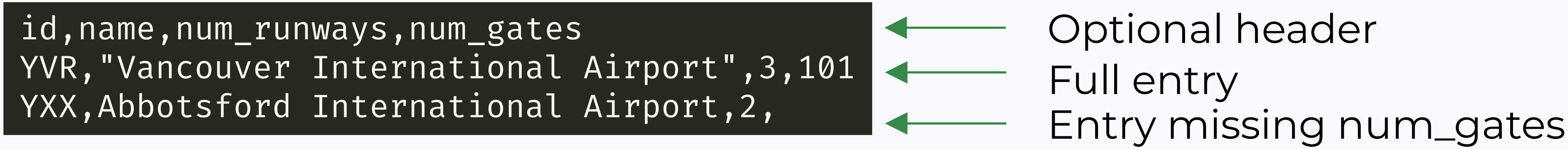
- There are two commonly used ways to store data

  1. CSV (Comma Separated Values)

  2. JSON (Javascript Object Notation)

CSV

- Good for simple data.
- Created by separating values with a comma.



```
id,name,num_runways,num_gates
YVR,"Vancouver International Airport",3,101
YXX,Abbotsford International Airport,2,
```

Optional header
Full entry
Entry missing num_gates

PCC 330-341

Reading CSV files

- Import csv module and create a csv reader with a file object.

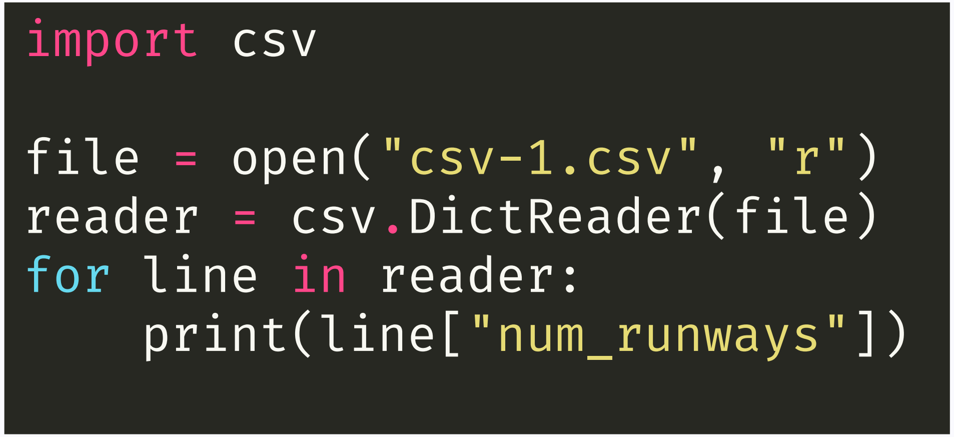
```python
import csv

file = open("csv-1.csv", "r")
reader = csv.reader(file)
header = next(reader)
for line in reader:
    print(line)
```

← Create csv reader
← Header first line (if exists)
← For all the other lines
← Do something here

- But for the **line** you *need to know* which index the data is.

Use DictReader

- Same as regular reader but will have each line be a dictionary instead of list with the keys as the header (or anything else you specify).



```python
import csv

file = open("csv-1.csv", "r")
reader = csv.DictReader(file)
for line in reader:
    print(line["num_runways"])
```

Create csv dictionary reader

For all lines

Do something here

Similarly can write

```python
import csv

file = open("csv-1.csv", "w")
writer = csv.writer(file)
writer.writerow(["id", "name", "num_runways", "num_gates"])
writer.writerow(["YVR", "Vancouver Intl Airport", 3, 101])
```

```python
import csv

file = open("csv-1.csv", "w")
fieldnames = ["id", "name", "num_runways", "num_gates"]
writer = csv.DictWriter(file, fieldnames=fieldnames)
writer.writeheader()
writer.writerow({"id": "YVR", "name": "Vancouver Intl
Airport", "num_runways": 3, "num_gates": 101})
```

Let’s practice

Create a class **Weather** which has a constructor that takes a path to the daily temperature CSV file and loads the data. Create a few functions:

- **average_high**: which returns the average high (deg C).
- **average_low**: which returns the average low (deg C).
- **average_max_gust**: which returns avg max gust (km/hr).

All functions optionally take **month** to specify for a month.

JSON

- Good for complex structured data.
- Javascript Object Notation which features a Python dictionary-like structure.

```
{
    "airports": [
        {
            "id": "YVR",
            "name": "Vancouver Intl Airport",
            "num_runways": 3,
            "num_gates”: 101,
        }
    ]
}
```

Write with json dump

```
import json

airports = [
    {"id": "YVR", "name": "Vancouver Intl
Airport", "num_runways": 3, "num_gates": 101}
]

file = open("data.json", "w")
contents = json.dump({"airport": airports},
file)
file.close()
```

Read with json load

```python
import json

file = open("data.json", "r")
contents = json.load(file)
print(contents)
file.close()
```

```
{'airport': [{'id': 'YVR', 'name': 'Vancouver Intl Airport',
'num_runways': 3, 'num_gates': 101}]}
```

Let’s practice

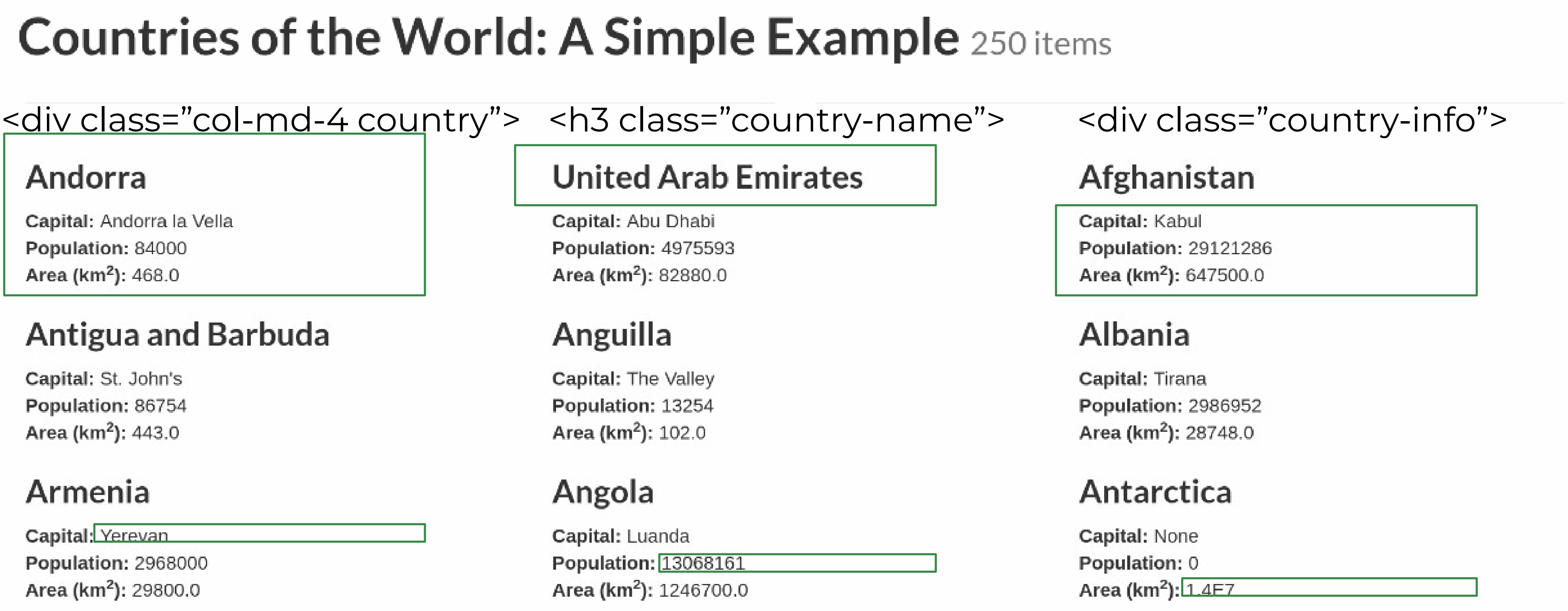
Create a class **Client** which has a constructor that will ask for the user’s first name, last name, date of birth, and phone number if it has not already asked and the data file does not exist. If exists load the data to the right attributes.

Web Scraping

Not all data are in files

- In this case we need to use web scraping to get and save web data.
- Website data is stored as HTML which we can download and process with the help of two modules: requests (downloading) and beautifulsoup4 (processing).
- We will learn more about HTML later.

Find the tags, classes, ids



Countries of the World: A Simple Example 250 items

<div class=”col-md-4 country”> <h3 class=”country-name”> <div class=”country-info”>

Andorra

Capital: Andorra la Vella
Population: 84000
Area (km^2): 468.0

United Arab Emirates

Capital: Abu Dhabi
Population: 4975593
Area (km^2): 82880.0

Afghanistan

Capital: Kabul
Population: 29121286
Area (km^2): 647500.0

Antigua and Barbuda

Capital: St. John's
Population: 86754
Area (km^2): 443.0

Anguilla

Capital: The Valley
Population: 13254
Area (km^2): 102.0

Albania

Capital: Tirana
Population: 2986952
Area (km^2): 28748.0

Armenia

Capital: Yerevan
Population: 2968000
Area (km^2): 29800.0

Angola

Capital: Luanda
Population: 13068161
Area (km^2): 1246700.0

Antarctica

Capital: None
Population: 0
Area (km^2): 1.4E7

<span class=”country-capital”> <span class=”country-population”> <span class=”country-area”>

```
response =
requests.get("https://www.scrapethissite.com/pages/simple/")
soup = BeautifulSoup(response.text, "html.parser")
countries = []
for country in soup.find_all("div", "country"):
    country_name = country.find("h3", "country-
name").get_text(strip=True)
    country_data = country.find("div", "country-info")
    country_capital = country_data.find("span", "country-
capital").get_text(strip=True)
    country_population = country_data.find("span", "country-
population").get_text(strip=True)
    country_area = country_data.find("span", "country-
area").get_text(strip=True)
    countries.append(
        { "name": country_name, "capital": country_capital,
          "population": country_population, "area": country_area,
    })

print(countries)
```

Let’s practice

Create a class **Hockey** which has an empty constructor which will scrape the site: https://www.scrapethissite.com/pages/forms/. Your goal is to take all the data and have it loaded into a python list and save as an attribute.