CEGE0096: Geospatial Programming

Lecture 3: Python Data Structures, IDE and Debugger

October 14, 2019

Aldo Lipani

PYTHON III

Our Python (Imperative) Journey

- Types
- Operators
- Variables
- Simple I/O
- Branching

} Covered in the 1st Lecture

- Loops
- Functions
- Program Planning

} Covered in the 2nd Lecture

- Strings
- Tuples
- Lists
- Dictionaries
- Files

Review Exercise 3

Write a program that **keeps requesting** the user to `input` a temperature in degree Celsius and `print`s out a suitable message according to the temperature state below:

- "Freezing" when the temperature is below zero;
- "Very cold" when it's between 0 and 10 (not included);
- "Cold" when it's between 10 and 20 (not included);
- "Normal" when it's between 20 and 30 (not included);
- "Hot" when it's between 30 and 40 (not included);
- "Very hot" when it's above 40.

The program should stop when the user enters a number greater or equal than 100.

You are allowed to use variables, operators, if statements and loops.

Function reminder:

`input("message")` prints the message and waits the user for an input

`float` to convert strings to floating-point numbers

Solution

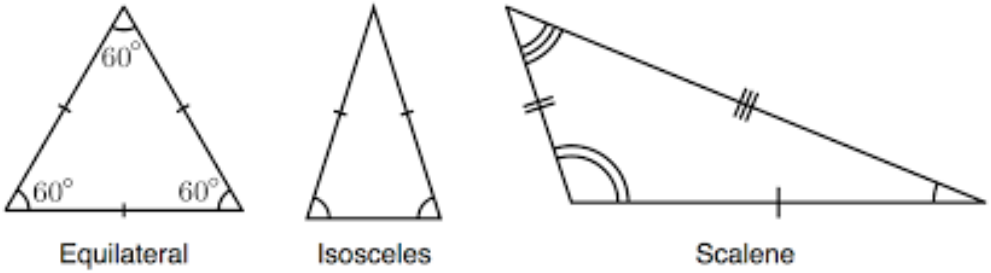
```
print("Temperature to Perceived Temperature Converter (0.2)")

temp = input("What temperature is outside?")
temp = float(temp)

while temp < 100:
    perceived_temp = ""
    if temp < 0:
        perceived_temp = "freezing"
    elif temp < 10:
        perceived_temp = "very cold"
    elif temp < 20:
        perceived_temp = "cold"
    elif temp < 30:
        perceived_temp = "normal"
    elif temp < 40:
        perceived_temp = "hot"
    else:
        perceived_temp = "very hot"

    print("Your temperature today is", perceived_temp)

    temp = input("What temperature is outside?")
    temp = float(temp)

print("Huuu, with such heat I better shut myself down!")
```

Changes with respect to the solution of Exercise 1 are in **bold**!

Review Exercise 4

Write a program that ask the user for 3 angles in degrees and checks, if possible, which triangle you can build with them:

- Equilateral triangles have all angles equal;
- Isosceles triangles have 2 angles equal;
- Scalene triangles have all of them are different.



This check should be done in a function. The function should return the type of triangle.

Rules are the same of the previous exercise.

Solution

```
def get_triangle_type(angle1, angle2, angle3):
    if angle1 + angle2 + angle3 != 180:
        triangle_type = "none"
    elif angle1 == angle2 and angle2 == angle3:
        triangle_type = "equilateral"
    elif angle1 == angle2 or angle2 == angle3 or angle3 == angle1:
        triangle_type = "isosceles"
    else:
        triangle_type = "scalene"
    return triangle_type

print("Angles Checkers for Triangles (1.1)")

angle1 = float(input("1st angle: "))
angle2 = float(input("2nd angle: "))
angle3 = float(input("3rd angle: "))

triangle_type = get_triangle_type(angle1, angle2, angle3)

if triangle_type == "none":
    print("You can't build a triangle out of these angles!!!")
else:
    print("You can build an", triangle_type, "triangle")
```

So far

Can you write a program that averages a series of user's inputs? Yes!

```
n = int(input("How many values?"))

num = 0
for i in range(n):
    v = float(input("Value number " + str(n) + ": "))
    num = num + v

print("The average is:", num / n)
```

Can you compute this average again without the 2nd element?

Umh!

Data Structures

In Python there exists 4 data structures:

1. Lists
2. Tuples
3. Sets
4. Dictionaries

These should be used based on your needs. Usually also to maximize the overall speed of your program in performing these operations:

1. Access
2. Search
3. Insert
4. Deletion

Tuples ()

Tuples are **immutable** sequences of objects.

You declare them by using `()` and separating the objects by commas.

```
empty_tuple = ()
another_tuple = (1, 2, 3, "hello")
```

To know the length of a tuple you can use `len`.

You have already seen them when returning more than one value in functions:

```
return (a, b)
```

You can access the elements using the slicing operators (like strings):

```
print(tuple[2:4])
print(tuple[1])
print(tuple[-1:-3:-1])
```

Lists [ ]

Lists are like tuples but they are **mutable**.

```
empty_list = []
another_list = [1, 2, 3, "hello"]
```

To know the length of a list you can use `len`.

You can access lists values like for the tuples using the slicing operators.

You can change the values of lists like:

```
another_list[2] = 5
```

Warning: make sure you have more than 2 values in the list, or list index out of range error.

Lists [ ] (II)

You can **append** a value to a list:

```
a_list.append("world")
```

You can insert a new value before a given value:

```
a_list.insert(0, "hello")
```

You can remove (and return) an item from the list at a given index (if no index is specified the last element will be removed):

```
a_list.pop(3)
```

And many more.

Sets {} (set() if empty)

A set is an unordered collection of objects with no duplicates.

```
empty_set = set()
another_set = {"apples", "oranges"}
```

To know the length of a set you can use `len`.

Like lists, sets are **mutable**. You can `add`, `remove` (`discard`), or `pop` an element. The `pop` is done on an arbitrary element.

To check if an element belongs to a set you can use the `in` operator:

```
"pears" in another_set
```

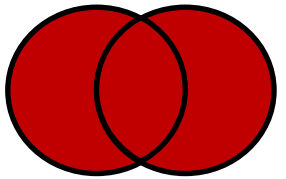
Where have we seen the `in` operator before?

Sets {} (II)

Between two sets you can perform sets operations.

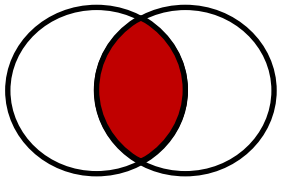
Union between two sets:

```
set1.union(set2)
```



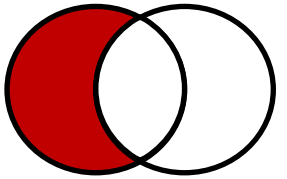
Intersection between two sets:

```
set1.intersection(set2)
```



Difference between two sets:

```
set1.difference(set2)
```



And many more.

Dictionaries {}

Dictionaries are mutable and unordered collections of key - value pairs. Like an actual dictionary you have for each word (keys) a definition (values).

```
empty_dictionary = {}
another_dictionary = {"apples":10, "oranges":2}
```

To know the length of a dictionary you can use `len`.

You can access an element like:

```
another_dictionary["apples"]
```

You can add or update an existing key:

```
another_dictionary["pears"] = 0
```

Dictionaries {} (II)

You can access the set of keys or values via the `keys()` and `values()` methods:

```
another_dictionary.keys()
```

```
another_dictionary.values()
```

To check if a key belongs to a dictionary you can use the `in` operator:

```
"pears" in another_dictionary
```

For Loops and Data Structures

You can iterate across a data structure using for loops. This will print in order the objects contained in `tuple_or_list`:

```
for value in tuple_or_list:
    print(value)
```

This will print the keys (or object for sets) contained in a `set_or_dictionary` (no order is guaranteed):

```
for value in set_or_dictionary:
    print(value)
```

For dictionaries, use `items()` if you need to iterate across keys and values at the same time:

```
for key, value in dictionary.items():
    print(key, value)
```

Strings (II)

It turns out that strings behave similarly to lists.

You can iterate across the characters of a string like:

```
string = "hello"
n = len(string)
for i in range(n):
    print(string[i])
```

```
string = "hello"
for c in string:
    print(c)
```

String Manipulation

You can split a string based on a given character like this:

```
string = "hello,world!"
string.split(",") # ['hello', 'world!']
```

You can can remove white spaces around a string using `strip`:

```
string = " world! "
string.strip() # 'world!'
```

You can replace substrings of a string using `replace`:

```
string = "hello,world!"
print(string.replace("e", "o")) # 'hollo,world!'
```

Performance of Data Structures

	Access	Search	Insert	Deletion
Tuples	**Fast**	Slow	N/A	N/A
Lists	**Fast**	Slow	Slow	Slow
Sets	N/A	**Fast**	**Fast**	**Fast**
Dictionaries	N/A	**Fast**	**Fast**	**Fast**

Access an element by index;

Search an element by value;

Insert an element in the data structure;

Delete and element from the data structure.

Which Data Structure

Which data structure would you use if you need…

…to keep the order of a sequence of values inputted by the user? `list`

…to check whether a sequence of values inputted by the user contains duplicates? `set`

…to count how many times an item is inputted by the user? `dictionary`

…to return two values from a function? `tuple`

…to compute the average of a sequence of values once? `a variable :P`

Files

You can open a file with the built-in function `open`.

Open takes two parameters, `file_name` (with path) and `mode`:

Use mode `"w"` (for write) to write the file like this:

```
string = "Hello, World!"
with open("file_name", "w") as f:
    f.write(string)
```

Use mode `"r"` (for read) to read the file like this:

```
with open("file_name", "r") as f:
    print(f.readline())
```

If you open a file using `with` you don't need to do anything else (like closing the file or handling exceptions).

Exercise 5 (30 Minutes)

Given a sequence of numbers inputted by the user compute the following statistics:

- min,
- max,
- average,
- number of duplicates, and
- mode.

You should define a function for each one of these statistics.

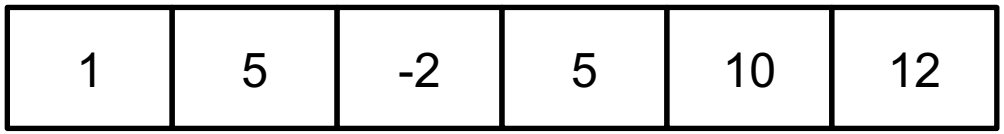
TIP: these functions should take as input a sequence of numbers.

Solution

```
print("Describe My Sequence (0.1)")

# functions go here

n = int(input("How many values? "))
l = []
for i in range(n):
    v = float(input("Insert number " + str(i + 1) + ": "))
    l.append(v)



print("The statistics are:")
print("* Min:", min(l))
print("* Max:", max(l))
print("* Avg:", average(l))
print("* Dup:", duplicates(l))
print("* Mod:", mode(l))
```

Solution (I)

```
def min(vs):
    res = vs[0]
    for v in vs[1:]:
        if v < res:
            res = v
    return res
```



Solution (II)

```
def max(vs):
    res = vs[0]
    for v in vs[1:]:
        if v > res:
            res = v
    return res
```

1	5	-2	5	10	12

Solution (III)

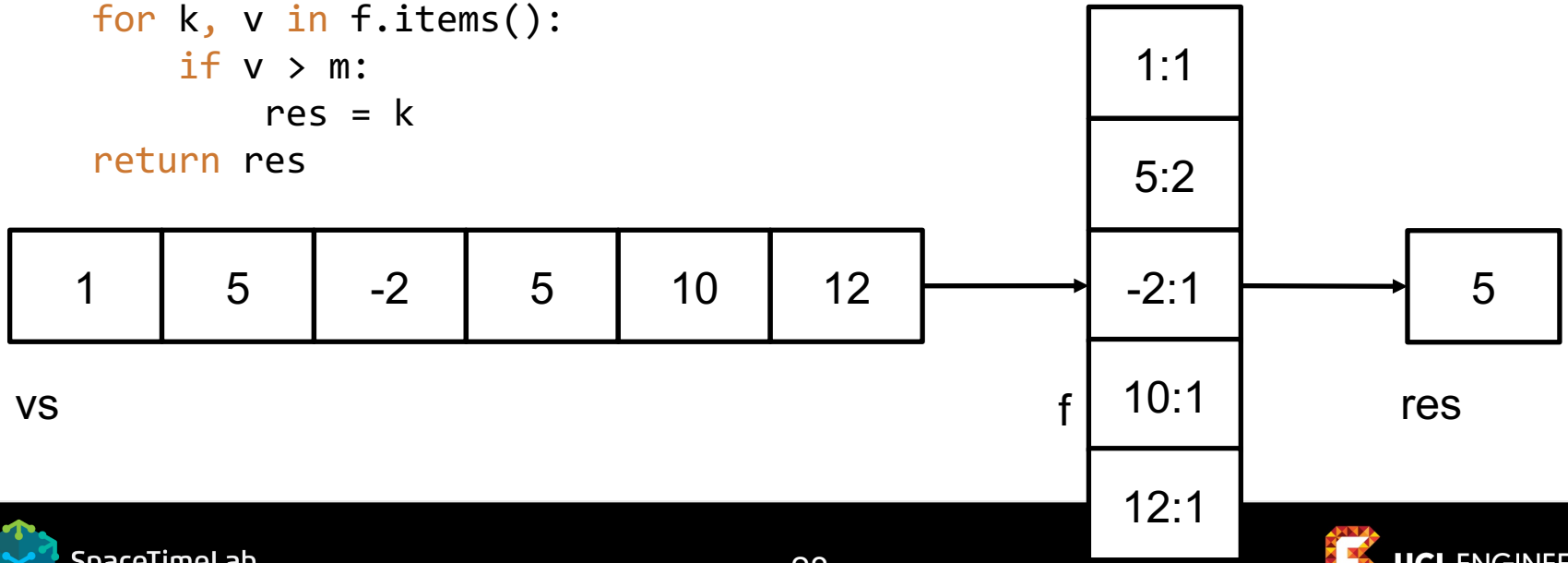
```
def average(vs):
    res = 0
    for v in vs:
        res = res + v
    res / len(vs)
    return res
```

Solution (IV)

```
def duplicates(vs):
    s = set()
    res = 0
    for v in vs:
        if v not in s:
            s.add(v)
        else:
            res = res + 1
    return res
```

Solution (V)

```
def mode(vs):
    f = {}
    for v in vs:
        if v not in f:
            f[v] = 0
        f[v] = f[v] + 1
    m = 0
    res = vs[0]
    for k, v in f.items():
        if v > m:
            res = k
    return res
```



Exercise 6 (10 Minutes)

Write a program that **asks** the user to `input` a temperature in degree Celsius and `print`s out a suitable message according to the temperature state below:

- "Freezing" when the temperature is below t_1;
- "Very cold" when it's between t_1 and t_2 (not included);
- "Cold" when it's between t_2 and t_3 (not included);
- "Normal" when it's between t_4 and t_5 (not included);
- "Hot" when it's between t_5 and t_6 (not included);
- "Very hot" when it's above t_7.

The temperatures $t_1, t_2, t_3, t_4, t_5, t_6$ and t_7 are given as input by the user.

IDE AND DEBUGGER

Anaconda

Python is an open-source programming language. Python is free to download from the Python Software Foundation.

Other organisations provide Python. For example, Esri distribute a version of Python that is compatible with ArcGIS.

Continuum Analytics distribute Python (and R) together with a bundle of data science libraries branded as **Anaconda**. Their business model is to sell support services to organisations that require operational reliability and security.

The Anaconda distribution system is installed with a package manager called **conda** and a GUI front-end called **Anaconda Navigator**.

Installing packages using conda provides a platform neutral solution and avoids many hours of hassle.

Anaconda (II)

You can define **environments**, like the `geospatial` one.

A conda environment is a directory that contains a specific version of Python and a collection of packages that you have installed.

Environments are **independent**: If you change one environment, the other environments are not affected.

You can easily **activate** or **deactivate environments**, which is how you switch between them.

You can also **share your environment** with someone by giving them a copy of your `environment.yaml` file.

Jupyter Notebook

Jupyter is an application that runs a Python kernel and provides input/output to that kernel via a **local webserver**.

The input/output is viewed using your favourite **browser**.

Jupyter provides a web-app in which you can create and edit `.ipynb` notebook files.

These files combine **markdown text** and **python code**.

Jupyter notebooks are used for:

- Developing small projects
- Documenting a workflow
- Providing training resources

Debugger

A debugger is a computer program that is used to test programs, and search for and eliminate errors in programs (aka **bugs**).

A debugger offers a thorough understanding of the workflow of a program, by providing:

- A step-by-step execution;
- A complete view of all the variables defined in the program;
- The ability to set breakpoints.

The most notable Python debugger is `pdb`. This is used like a library and coded in your program. It is flexible because it can be used in any python program. However, it is not the most user-friendly debugger.

`pdb` will not be covered in this module.

Integrated Development Environment (IDE)

An IDE is a software application that provides **comprehensive facilities to computer programmers for software development**.

They tend to be **language specific**. An IDE can be the best for one language but not as good for another.

They tend to include **all you need** to create a program in a given programming language, including a source code editor, build automation tools, and a **debugger**.

The boundary between an IDE and other parts of the broader software development environment is not well-defined; sometimes a version control system or various tools to simplify the construction of a graphical user interface (GUI) are integrated.

Some IDE allow the installation of third-party plug-ins to further extend their functionalities.

There are many IDEs for Python.

PyCharm (IDE)

PyCharm Community edition is a free open-source IDE with a smart Python editor.

PyCharm provides quick code navigation, code completion, refactoring, unit testing and debugger.

Professional edition fully supports Web development with Django, Flask, Mako and Web2Py and allows to develop remotely.

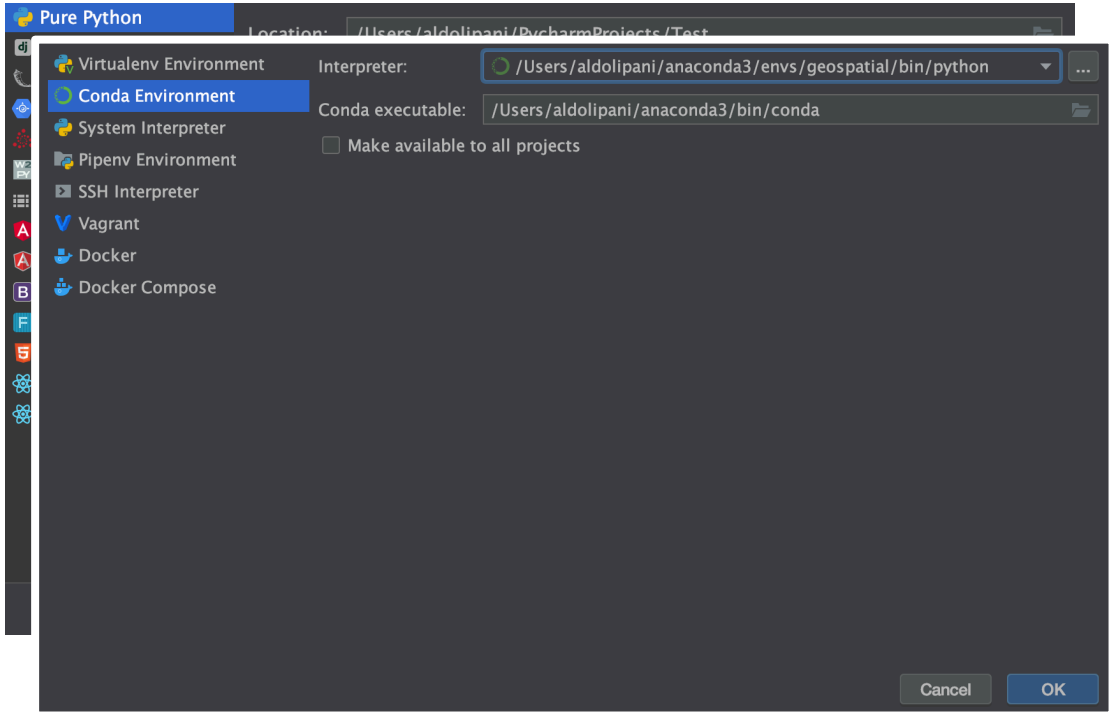
JetBrains offers free PyCharm professional licenses for Student/Educational use.

PyCharm Ecosystem

PEP8

PyCharm

File ↘ New Project



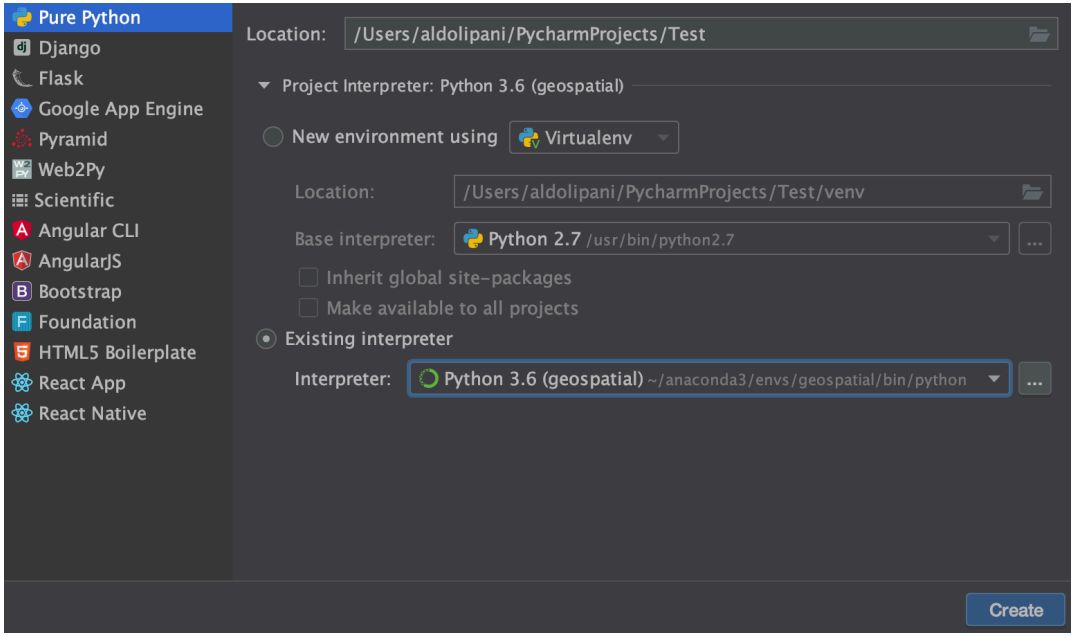
Name and Location of the Project

Python Interpreter

If the Anaconda geospatial env is not available, click here to select it.

PyCharm

File ↘ New Project

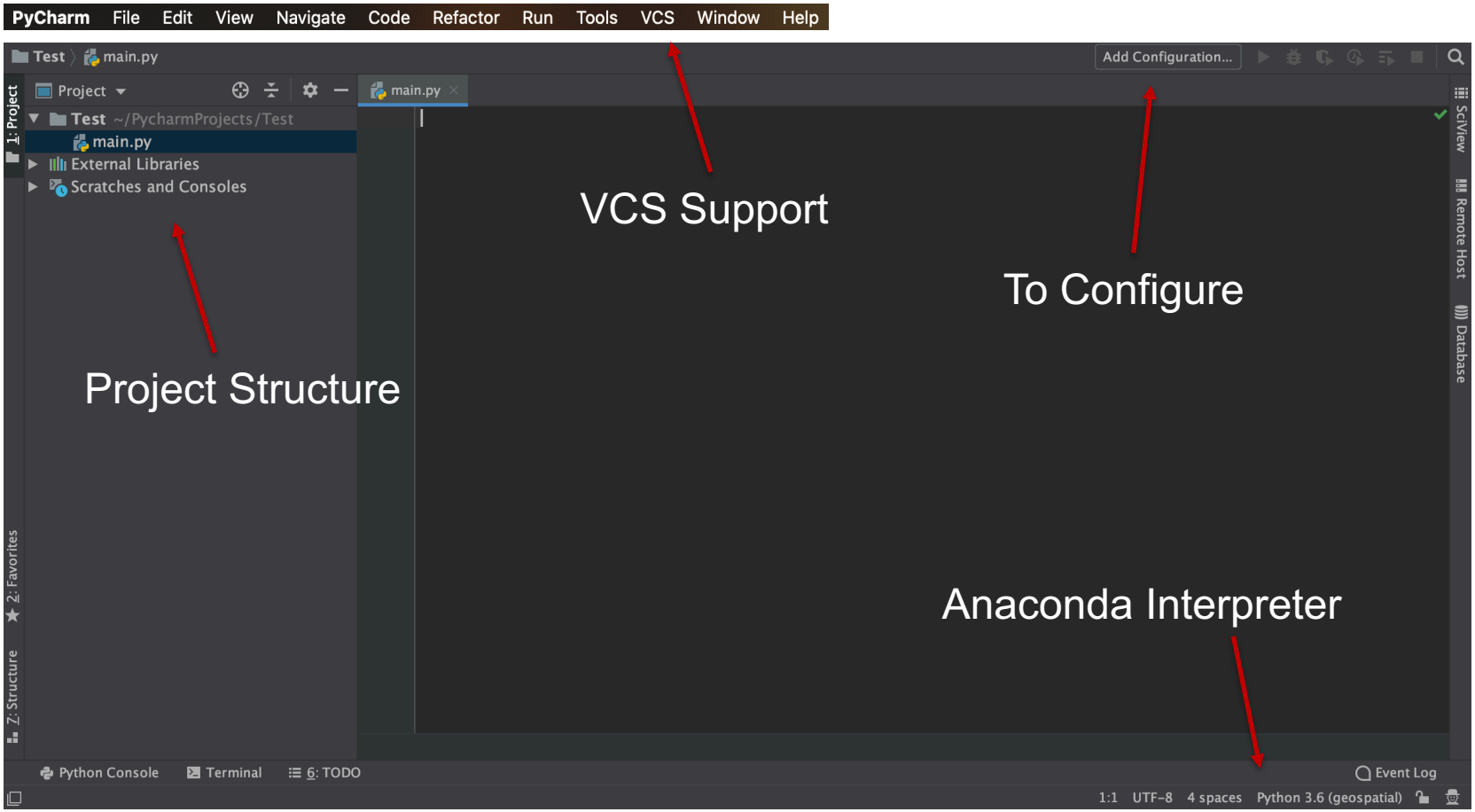


Name and Location of the Project

Anaconda Integration

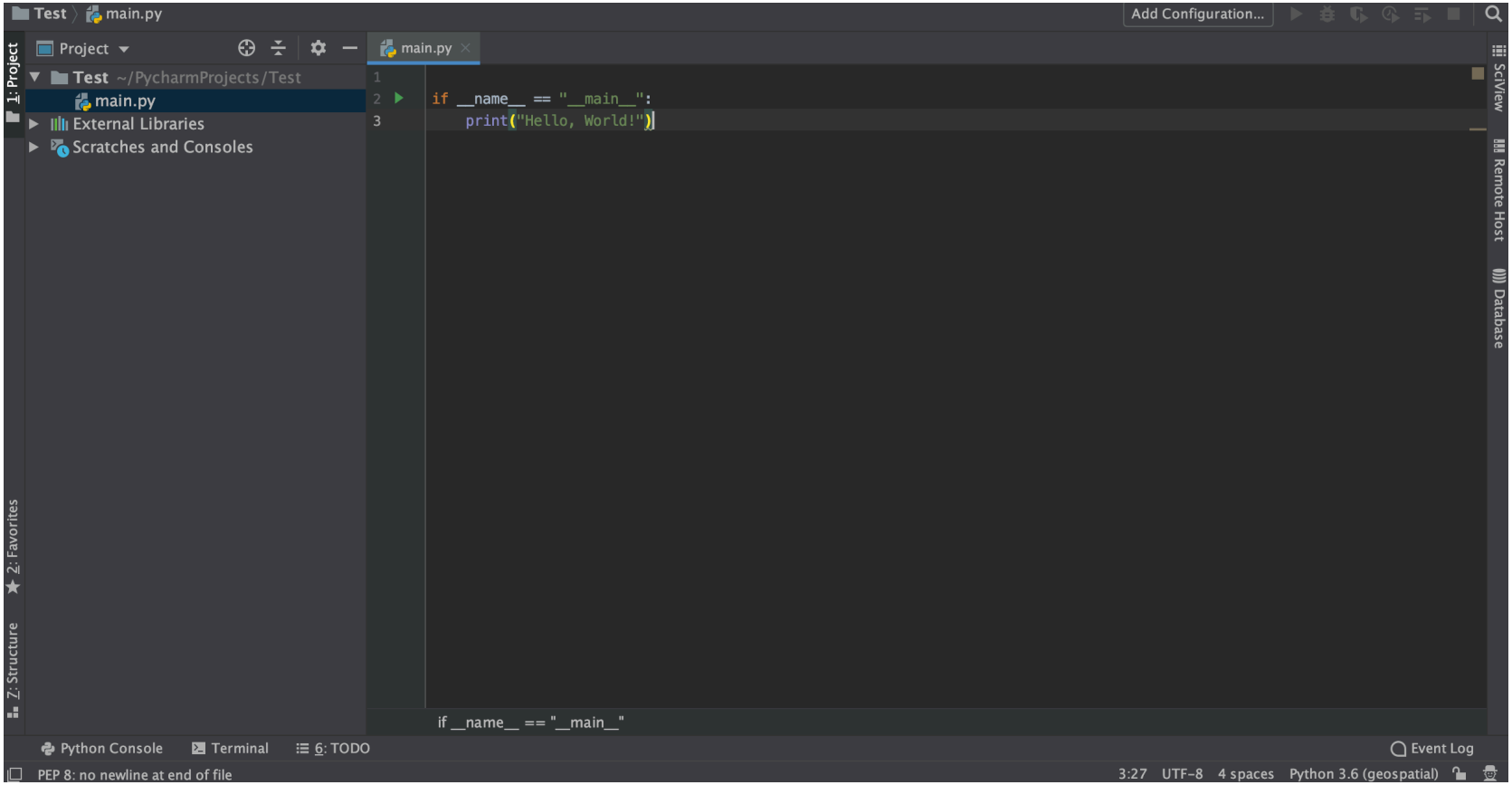
PyCharm

After having created a file `main.py`…



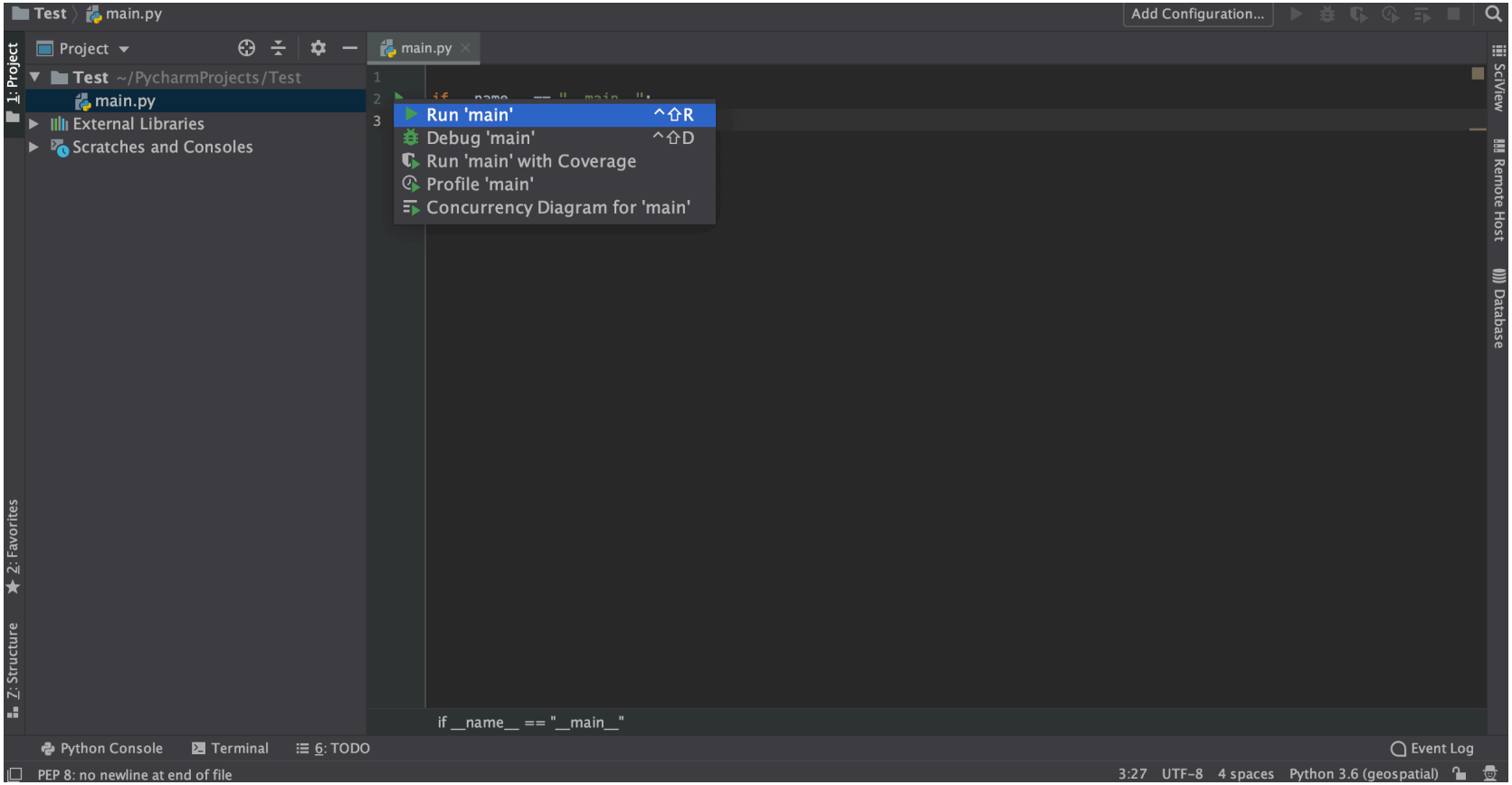
PyCharm

After writing an `if __name__ == "__main__":` a green play sign appears.



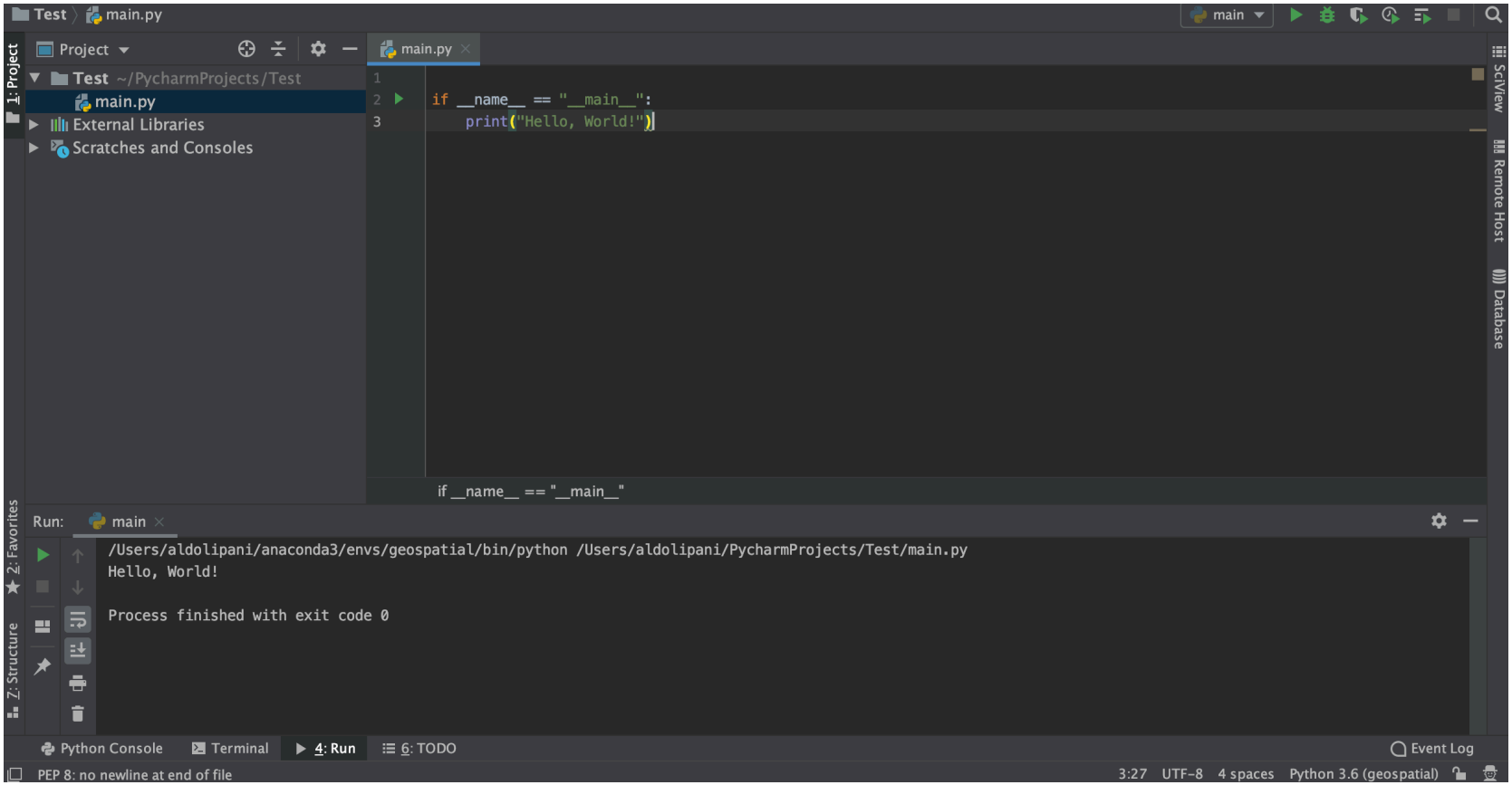
PyCharm

It will ask if you want to run the main file (Click on it).



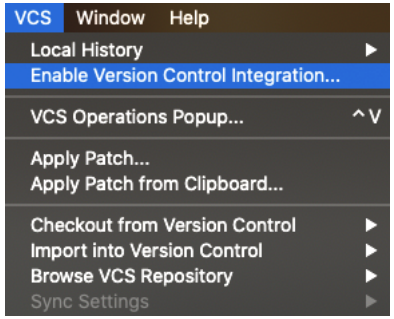
PyCharm

This will run the program and show the result in the Run Window below. Notice that the top-right dropdown menu is now configured to main and the other buttons are activated.

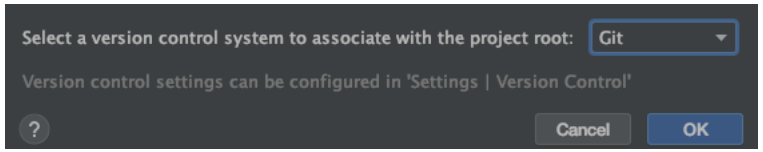


VCS in PyCharm

You first need to enable version control integration through the VCS menu:



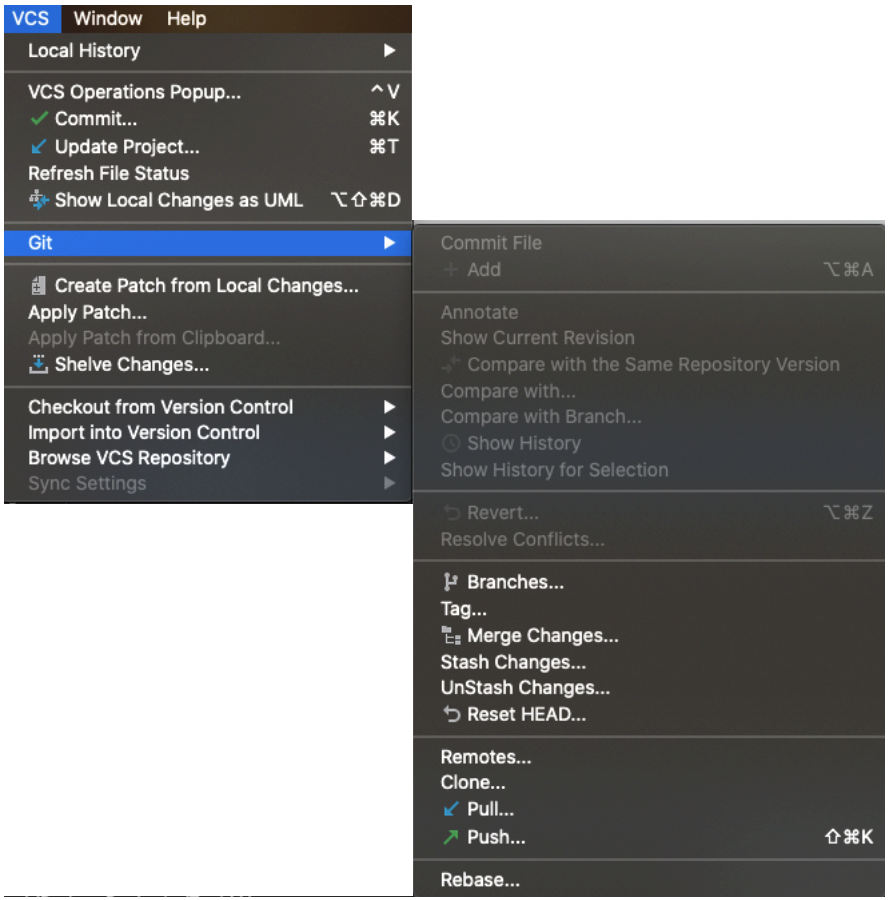
Then, you need to select Git:

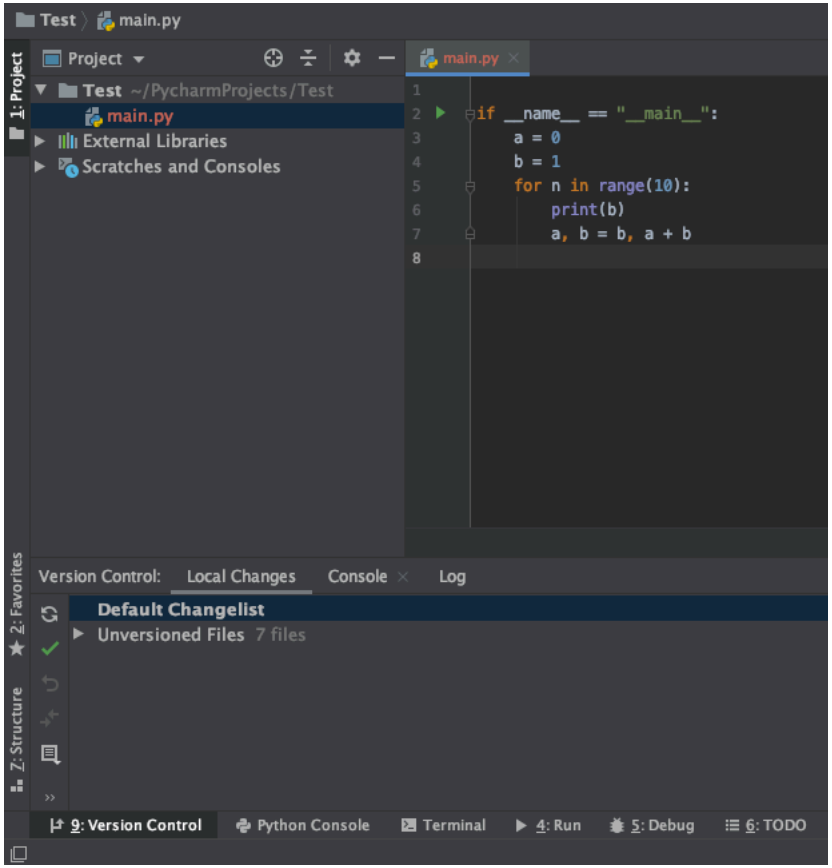


If Git was correctly installed in your machine then these steps should be enough to make it work.

VCS in PyCharm

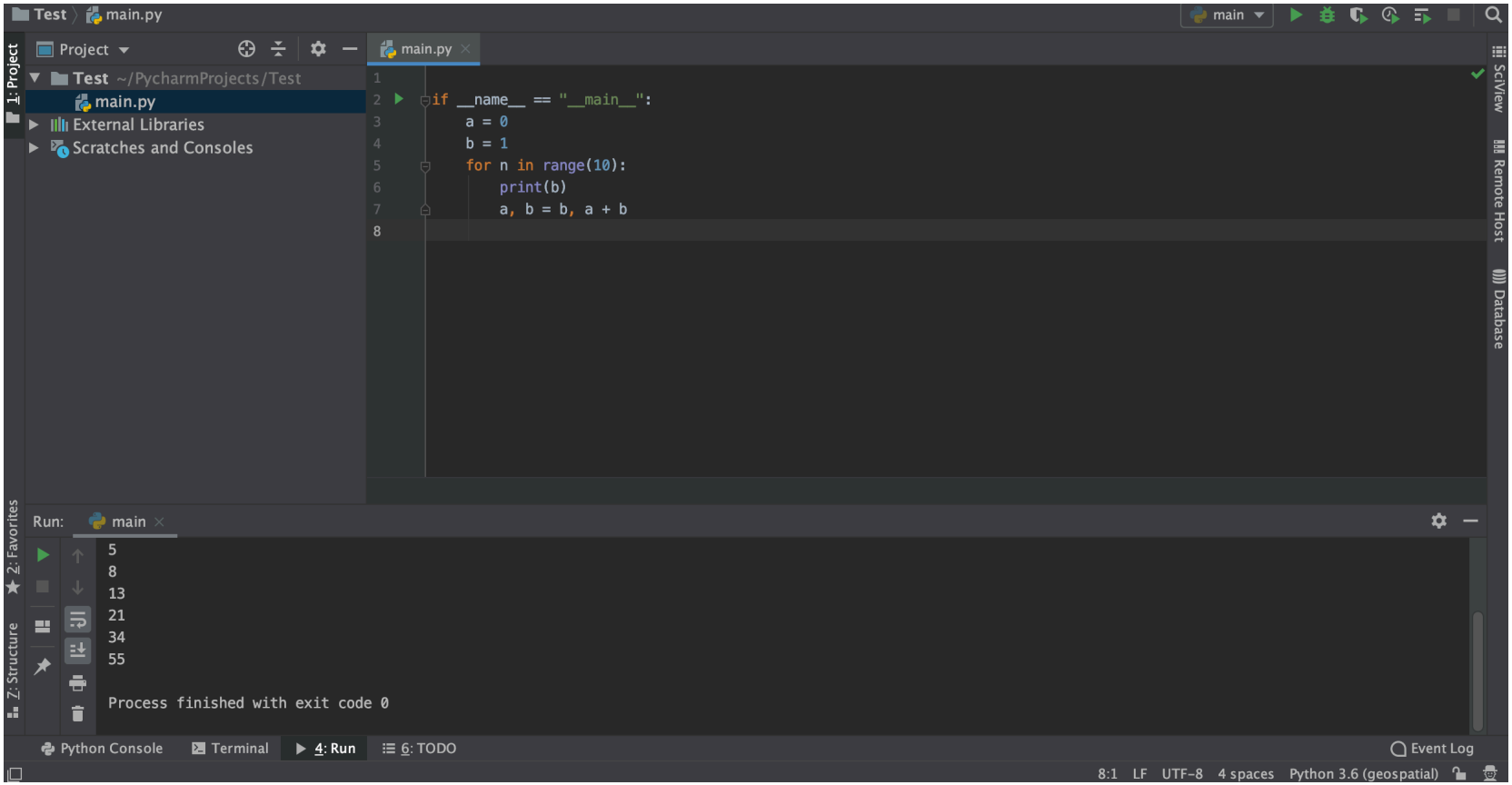
You should now have a new VCS menu and a new Tool Window, the Version Control Window.





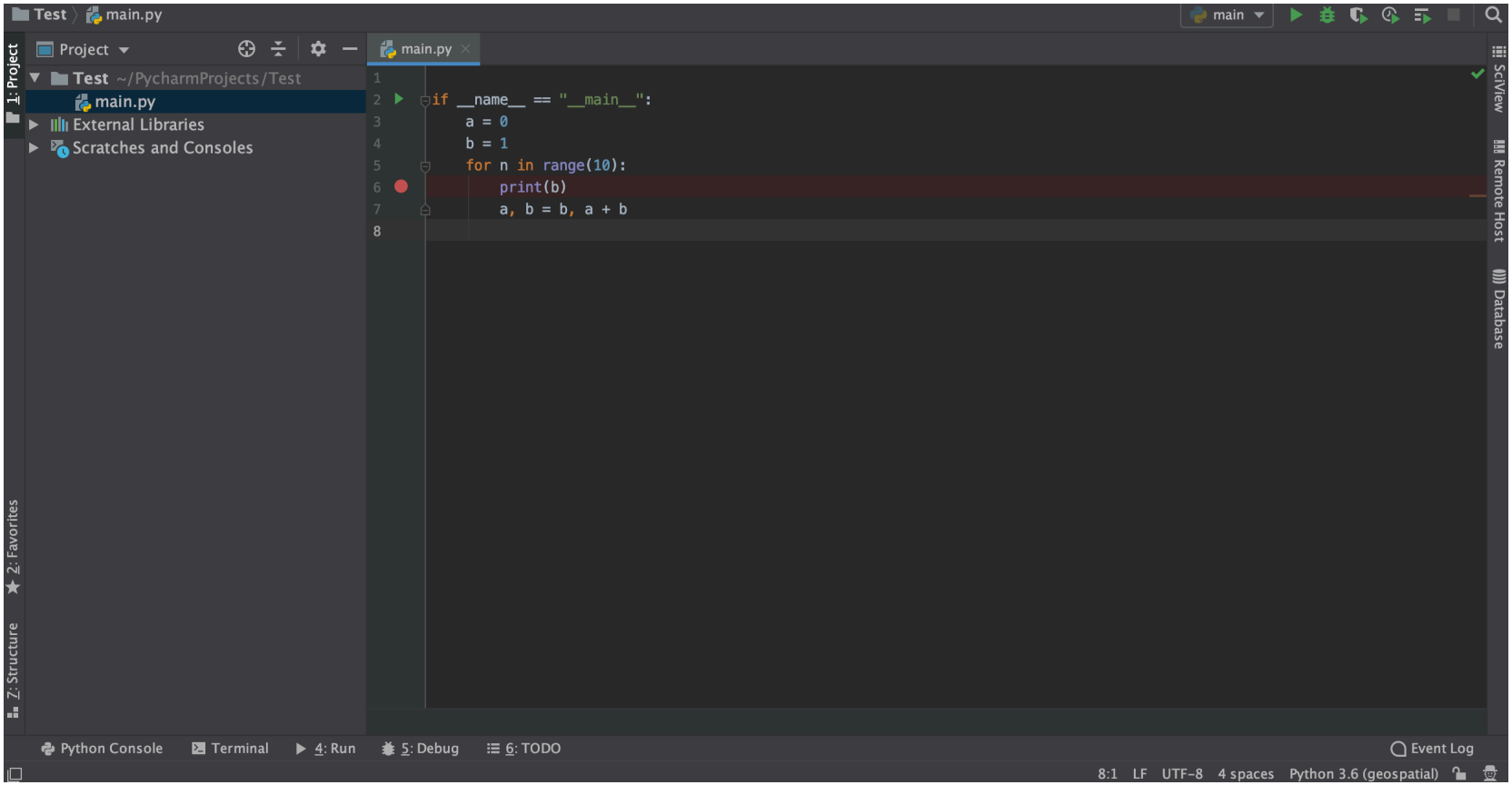
Debugging in PyCharm

The Fibonacci series



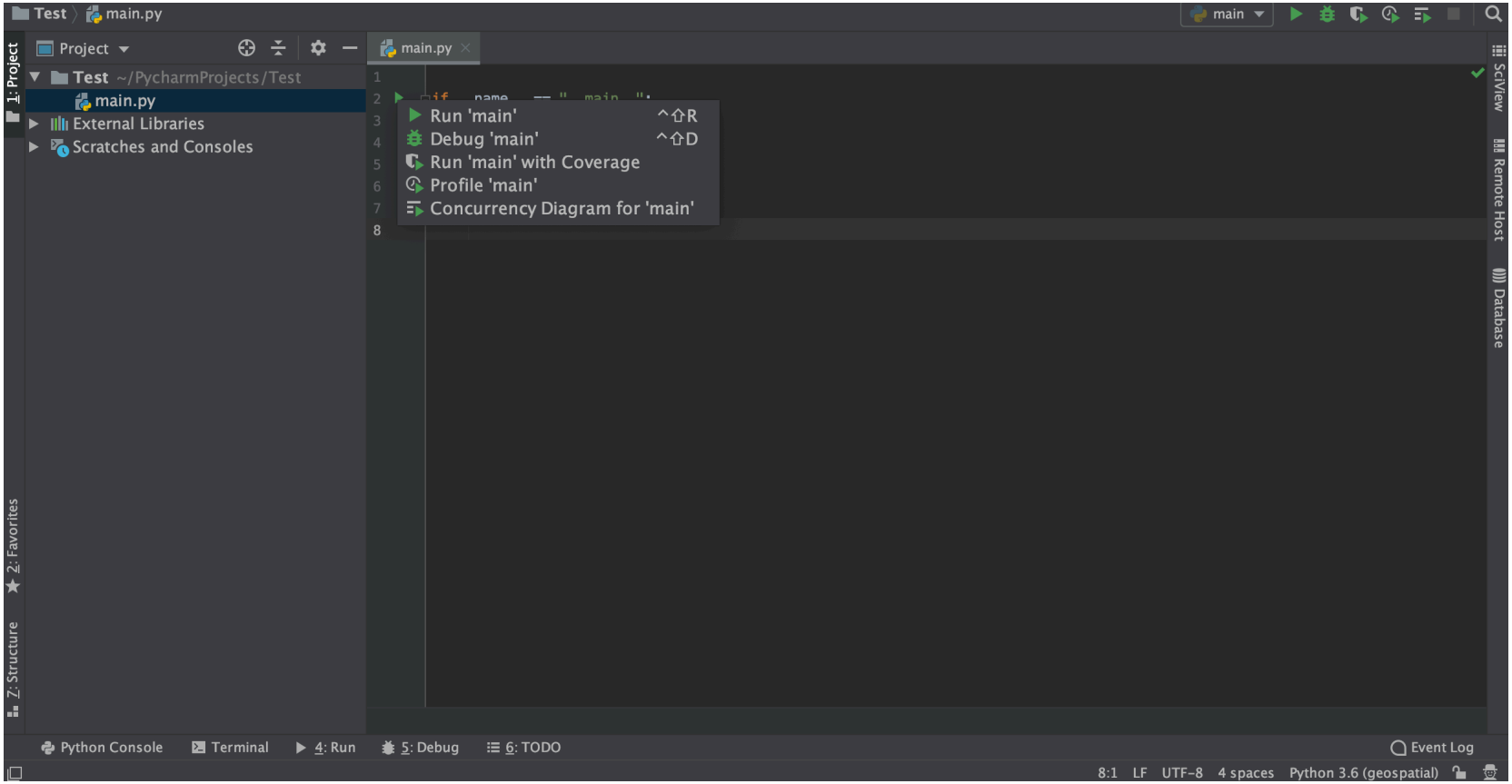
Debugging in PyCharm

Set a breakpoint wherever you want to stop the execution.



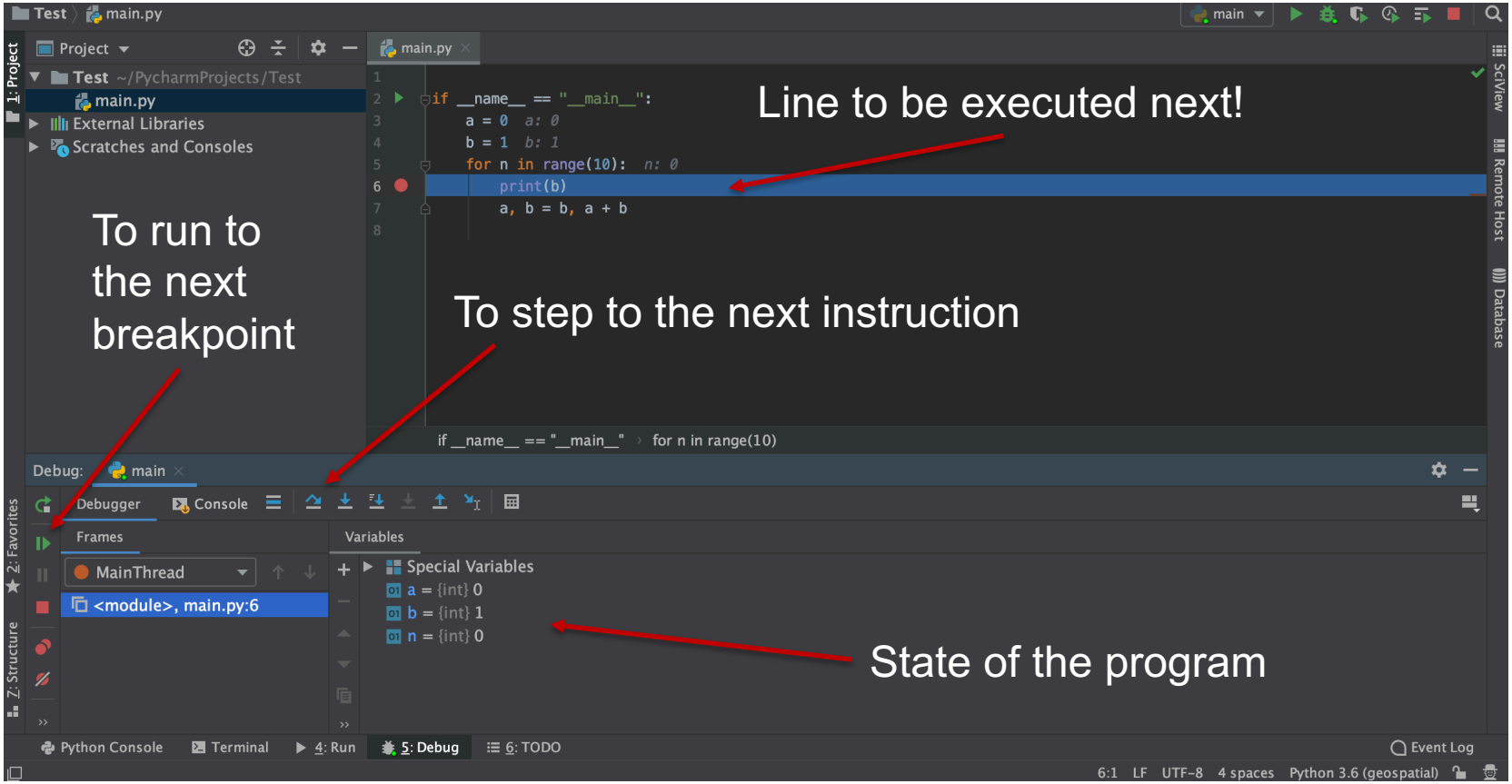
Debugging in PyCharm

Run the code in debug mode by clicking on the green bug icon (on the drop-down menu after clicking the green play button or directly from the top-right menu).



Debugging in PyCharm

You are now in debugging mode.



See you on Thursday for the Practical

Christopher Ingold Building G20

Do not forget to,

- install PyCharm on your personal laptop;
- take it with you for the next practical.