Python Basics

Learning outcomes

1. Create a Python file and run it
2. Do basic math operations with integer and floating point numbers
3. Do basic string manipulation, print formatted text
4. Create and manipulate variables, print numbers
5. Get user input from the terminal
6. Use comments in your code

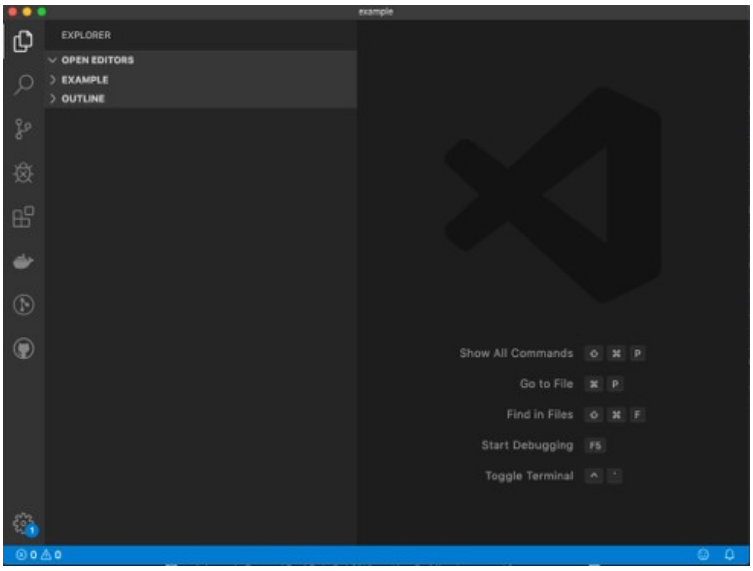
Preparation for today’s session

In the pre-course work, you can find instructions for how to install Anaconda, in which you can create your Python environment, and Visual Studio Code, which is a text editor for code. Please make sure you've done this before the first class, because we're going to dive right into the exercises. If you're having trouble installing anything, please email your course lead to let them know (see your welcome email for their email address).

Let's get started!

Open your first project

After you have installed VSCode, you should have a shortcut to open it. That shortcut will be placed at a different location depending on your operating system, e.g. for Windows it can be found in your Start Menu and for macOS you can find it in your Spotlight Search. Find the shortcut and open VSCode, and when you see the Welcome screen, find the window menu and click on File -> Open... Now navigate to an empty folder on your computer you want to use as a folder for your first Python project and click Open. You will see that the look of VSCode will change and an explorer column (pane) on the left of the program window will appear. This is where you will be able to see all your project files once you start creating them.



1. Create your first Python file

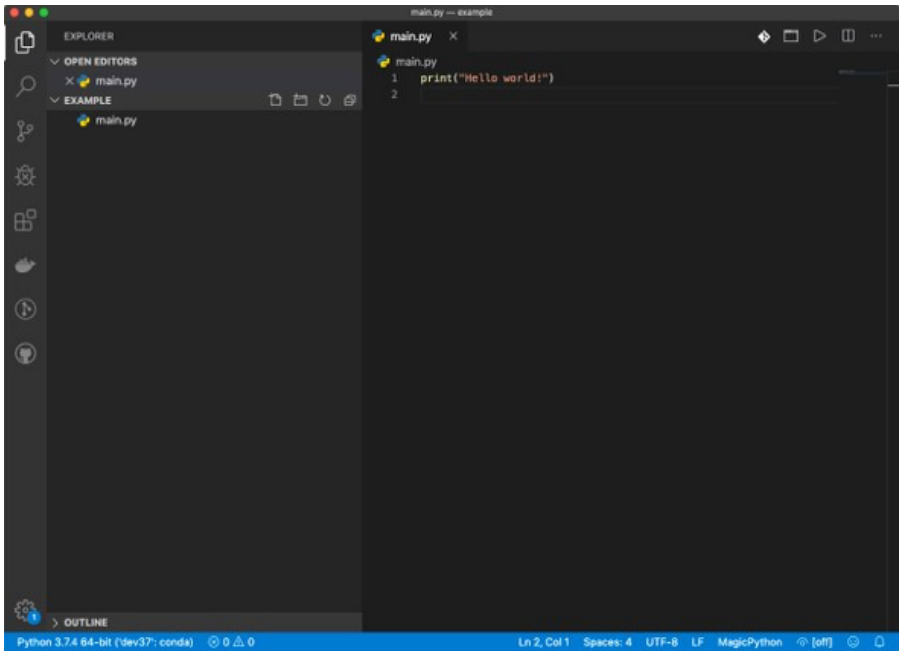
Opening an empty folder will show an empty Explorer pane. Now let's create your very first Python program!

1. Expand the second element in the Explorer pane ("EXAMPLE" in the screenshot)

2. Right click and choose "New File"
3. VSCode will ask you for the filename (for now you can just call it `main.py`)

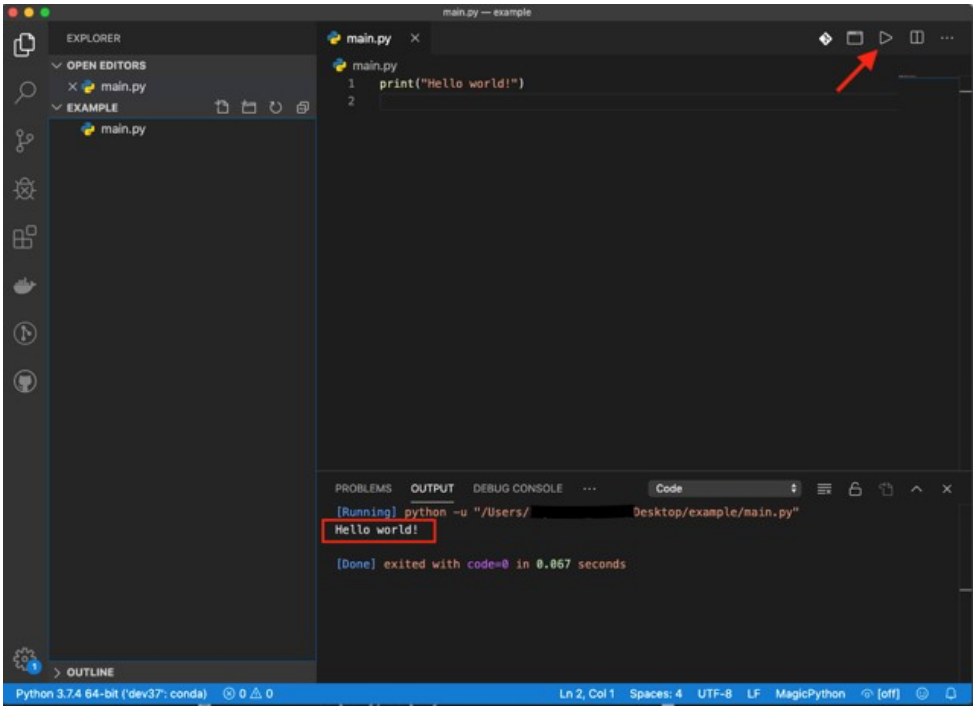
All Python programs have the extension `.py` and that is how code editors like VSCode know that the given file is a Python program, so whatever name you decide for your Python files, you need to make sure they end with `.py` . Now your Explorer pane will contain your file and if you click on it, VSCode will open it in the main pane of the window, also known as the Editor pane. This is where you are supposed to edit your files and type your programs in.

Enough talk, let's create your very first program! Let's type in the first line of the file `print("Hello world!")` . Well done! Now you have completed the very first step of becoming a Python programmer - this is the program which almost every Python programmer in the world started with!



Running your first Python program

After you've opened your very first Python file, you might have noticed that the bottom blue bar in the VSCode window has changed and is showing a Python version. You can now click on that Python version and choose the Python environment you have created in the pre-work. After you have selected your Python version, you can find the green play button in the top right corner of the VSCode window and run your program! Voila, your first Python program was executed! And you can see its output in the bottom of the window, where a new pane was opened - that is called the Terminal pane. You should see that the second to last line of the pane says `Hello world!` and that is the output of your program. Try to experiment a bit with your first program and see if you can make it say `Hello World!` or maybe even say hello to you?



What is a Python program?

Python programs are text files which contain instructions for Python on each line. So in your first Python program you have instructed Python to "print" some text to the screen by using the function `print()` . If you ever need your program to print something to the screen, you can use `print()` - you just need to make sure that the thing you want to show on the screen should be between the two braces and if it is text, it should be surrounded by quotes, like this `print("I am a Python program")` . All lines in a Python program should start immediately with a Python instruction or just be empty (unless another Python rule says otherwise), similarly how all sentences in English start with a capital letter.

While humans can easily understand misspelled sentences, Python is not that clever and will tell you that it cannot understand the line. You can test this and try to confuse Python by modifying your first program, e.g. by adding a space symbol before `print` or maybe changing it to `print1` . What does Python tell you when you do that? Can you create a program which will "print" two things one after another on the screen?

2. Calculations in Python

Python is only one of many programming languages computers speak, and we all know that they also speak another language very well - the language of maths! You can program your computer to do maths with Python. Now, create another file `calculations.py` and open it for editing. Type in `print(1 + 1)` at the first line and run your file. What did Python show in your Terminal window? What about if you type `print("1 + 1 =", 1 + 1)` ?

Python supports many mathematical instructions like:

- `+` addition
- `-` subtraction
- `*` multiplication
- `/` division
- `//` integer division
- `**` exponentiation
- `%` modulus (remainder)

These instructions are called operators. Try some calculations on your own, here are some examples for inspiration:

```
5 - 6
8 * 9
6 / 2
5 / 2
5 // 2
5 % 2
2 * (10 + 3)
2 ** 4
```

Was there anything surprising in the results? Can you explain in your own words what the integer division `//` operator and the modulus `%` operator do?

3. Strings in Python

In the beginning of this session you were told that you have to put all text you want to print to the screen in quotes, e.g. `print("text to print")` . Did you try to print the text without surrounding it in quotes? What does Python tell you when you forget to surround the text in quotes? When Python executes a program it is expecting a list of instructions for your computer. However numbers and text are not really instructions, but data which needs to be processed by those instructions.

One example is the data used by the function `print` when you type `print(1)` or `print("Hello")` . For Python it is really easy to recognise that numbers are just data since there are no instructions named after numbers. However when you type in text it is a bit unclear whether you meant to type an instruction or simply text. In order to make that clearer, Python needs you to surround your text with quotes when you don't want it to be treated as instructions. To test that quickly, check what `print(print)` will show on the screen when executed. And what about `print("print")` ? What does Python tell you in the first case?

Text can be surrounded by either single quotes 'text' or double quotes "text" , but not a mix of both "text' . It doesn't matter which style you use but Python programmers in general pick one and stick to it. Text surrounded by quotes is called a string and that is how we will refer to it from now on in the session. Now try to print a few strings on your own. Which style do you prefer to use, double or single quotes?

String operations

Python supports some operations on strings which can be very handy. For example, if you want to join two strings into one, you can do "Hello " + "world!" which will result in the combined string "Hello world!" . Go ahead and try to print each of these with print() . Is there any difference in the output when you use the + symbol?

Here are a few more examples to try:

```
print("Good" + " " + "morning!") print(len("hi"))
print(len("hello")) print("Bob" * 3) print("Bob" + 3)
print("hello".upper()) print("GOODBYE".lower())
print("the lord of the rings".title())
```

Did all the operations work? Was there a result that was surprising to you?

The examples here introduce you to a new function called len() . What do you think this function does after you have seen its output when used on different strings?

You can also see that we have introduced a new type of instruction here which looks like a function attached to the string with . like .upper() . These instructions are called methods and strings have a few of them you can use. What do you think .upper() , .lower() and .title() do? You can find more string methods here.

String formatting

When using strings in Python, you might need to create almost the same string but with one or two different words. To accommodate for that Python has a special function on strings which gives you the ability to create a template string and then fill in the template with the words or characters you want. To see how you can create a template string and use it, experiment with running the following program:

```
example = "My name is {} and I like {}!" print(example.format("Jessica", "pizza"))
print(example.format("John", "pineapple"))

# You can also do this instead
example = "My name is {0} and I like {1}!" print(example.format("Jessica", "pizza"))
print(example.format("John", "pineapple"))

# Why is the text different here? How would you fix it without changing the template?
example = "My name is {1} and I like {0}!" print(example.format("Jessica", "pizza"))
print(example.format("John", "pineapple"))
```

Sometimes, when you're doing exercises or looking at code online, you might see string formatting done with % instead of {} . This is old style formatting and it is done like this:

```
example = "My name is %s and I like the number %d!" print(example %
("Thea", 4)) print(example % ("John", 5))
```

But in general when you see %, it just means this:

```
%s → {}
%d → {}
%r → {}
```

If you're wondering what the letters stand for, s is for string, d is for decimal, and r is used to format a string in a particular way (you can read more about differences between old and new style formatting here). There's even a newer way (Python versions 3.6 and newer) of achieving string

templating by using f-strings and you can learn more about that here. However, it is best to use {} to make sure your code works across most Python versions.

4. Values, types, variables

As discussed earlier, Python makes a difference between data and instructions. Data in Python and other languages has value, e.g. 1, "Hello", 42 - these are all values. Each of those values has a type, e.g. the value "Hello" has a type string and the value 1 has a type integer. Some of the basic types in Python are:

- bool , which can have the value of either True or False int , which can have the value
- of any integer number like 1, 42, -456 float , which can have the value of any floating
- point number like 3.14, 0.1 str , which can have the value of any text like "Hello
- world!"

Python has even more basic types and if you are curious you can read more about them here.

Often it is very useful to think about the operations you want to perform on some values, before even knowing the values. For example, let's say we want to calculate our body mass index (BMI). After a quick lookup in Google we can find that the formula is

- BMI = weight / (height x height)

in the metric system, i.e. weight in kilograms and height in metres. So we can easily do print(80 / (1.80 * 1.80)) and get the BMI for a person who weighs 80 kilogram and is 1.80 meters tall. However, it is not very clear what this is calculating and what each number represents. Moreover, if another person wants to know their BMI, we need to modify all the numbers manually and we might easily forget to modify their height twice to get the right number. Python deals with this by using variables, which are essentially names for the values. So the formula would be print(weight / (height * height)) . But how do we tell Python that the weight is 80 and the height is 1.80? To do that, we use the assignment operator = before using the variables to assign values to them.

```
weight = 80
height = 1.80 print(weight / (height * height))
```

Now the code looks much clearer and we can understand the formula much better. We can even change the height only once and get the correct result when we run our code. You can imagine that variables are cardboard boxes, values are the content of those boxes and your program is a conveyor belt with places for those boxes. Your conveyor belt will always do the same thing to the boxes, no matter what they contain. Therefore you can use the same conveyor belt for any thing you want to transport.

You can change the values of your variables at any point in your program and your variables can even be assigned values of different types like:

```
highest_peak = "Everest"
print("The highest mountain is", highest_peak) highest_peak = 8848 print("The highest
mountain is", highest_peak, "metres high")
```

It might seem obvious, but it's worth pointing out that = is an "assignment operator". This means "set the name on the left equal to the value on the right". It isn't the same equals you see in maths!

In Python, it is convention for variable names to start with a lowercase letter. If you want to use multiple words in a name, you can separate them with an underscore, like this a_long_variable_name = "hello" . The operations you can perform on each variable are the same operations you can perform on the values contained in them as you can test with this small program:

```
age = 30
first_name = "Jessica" last_name = "Jones"

print(first_name + " " + last_name) print(first_name + " " + age)
```

Did Python execute all the lines of this program? If not, do you understand why?

Converting between types

Python supports an easy way for programmers to convert values between some of the types. For example, if you have a string with a number "100" and if you want to make it an int , you can simply use this in your program int("100") . You can also convert any other type to string by wrapping the value with str() , e.g. str(3.14) . To understand better how type conversions work, try to print a few examples on your own, using print() . Here are some examples to start with:

```
str(3.14) bool("")
bool(0) int(True)
float(10) int(7.6)
float("3.14")
int("Hello")
```

Did all the conversions work? Was there a conversion which was surprising to you?

We can now revisit a program you wrote earlier and could not be executed in full:

```
age = 30
first_name = "Jessica" last_name = "Jones"

print(first_name + " " + last_name) print(first_name + " " + age)
```

Can you think of a way to make this program complete without an error?

5. User input

Using variables in your code can help you create more complex programs. But what if you want the user of your program to provide the values for some of your variables? You cannot expect them to know how to code in Python in order to use your program, can you? So there must be a way for you to ask the user to input some values. Python provides such a function called input() .

Now create the following program and run it:

```
name = input("What is your name?") print("Hello {}! It's nice to meet
you!".format(name))
```

In your terminal window pane in VSCode you can see the question while the program stops and waits for you to enter your input. To do that, click in the terminal, type in your name and press Enter. Try running the program several times and provide different names. Does your program greet you in the way you expect it?

Could you think of a program which can calculate your BMI by asking you for your weight and height?

6. Comments

Reading Python programs is fairly easy compared to other programming languages. However it is still a bit harder than reading a book since it is not written in plain English and it takes time to understand what the programmers meant when writing the code. That's why programmers write comments in plain English so they can explain what a particular piece of code does. To distinguish code instructions from comments, all comments start with # . While they can be read by humans, Python ignores as if they don't exist.

Here are some comments in action:

```
# This line is a comment
greeting = "Hello World!" # This creates a variable greeting.upper() # This converts the string to
uppercase print(greeting * 3) # This prints the string 3 times
```

It is often a good idea to leave a comment for other programmers (or your future self) especially when a piece of code is doing a complex operation. You do not need to write comments in every single line if the code is easy to understand.

Zen of Python

Python is a powerful, open-source, and easy-to-learn programming language. Its design philosophy and syntax emphasise on code readability, and the ability to express concepts in fewer lines of code than with many other languages such as C++ or Java.

PEP 20 (The Zen of Python) summarises the language's core principles, which include:

- Readability counts
- Simple is better than complex
- Complex is better than complicated Explicit is
- better than implicit

As you can see, it's an ideal language for first-time programmers!

> *Easter Egg:* As a bonus, try running a new program that has this line at the top: `import this .`

Homework

1. Create a new Python file and create a program which prints the answers to these expressions:

```
10   / 3
10.0 / 3.0 10 % 3
10.0 % 3.0
```

2. Create a new Python file with the following statements in it, then run it. Were the things that were printed what you expected to see?

```
a = 1 a = a + 1
print(a) b = "hello"
print(b) c = b.title()
print(b) print(c) d =
"hello" e = d.title()
print(d) print(e)
name = "Dave"
f = "Hello {0}! ".format(name) print(f) name
= "Sarah" print(f) print(f * 5)
```

3. Work your way through exercises 1 to 8 on Learn Python The Hard Way.

Reference

- CFG Advanced Python course