Chapter 1

Introduction to Python

In this chapter we will discuss the very basics of Python. No we are not talking about snake anatomy, but the programming language Python. Python is a high-level, interpreted programming language known for its readability and versatility. It is widely used in various fields such as web development, data analysis, artificial intelligence, scientific computing, and more. It can be used for data parsing, visualization, and even machine learning. Python's syntax is designed to be easy to read and write, making it an excellent choice for beginners and experienced programmers alike.

1.1 Getting Started with Python

To get started with Python, you need to have it installed on your computer. You can download the latest version of Python from the official website: https://www.python.org/downloads/. Follow the installation instructions for your operating system.

Once you have python installed, you can write and run Python code using various methods:

- **Interactive Shell:** You can open a terminal or command prompt and type `python` or `python3` to start an interactive Python shell where you can type and execute Python commands line by line.
- **Script Files:** You can write your Python code in a text file with a `.py` extension and run it using the command `python filename.py` or `python3 filename.py` in the terminal.
- **Integrated Development Environments (IDEs):** There are several IDEs available for Python, such as PyCharm, VSCode, and Jupyter Notebooks, which provide a more user-friendly environment for writing and running Python code.

We are going to install VSCode as our IDE for this course. You can download it from https://code.visualstudio.com/. After installing VSCode, you will also need to install the Python extension for VSCode, which can be found in the Extensions Marketplace within VSCode.

1.2 Writing Your First Python Program

Let's start by writing a simple Python program that prints "Hello, World!" to the console. Open your IDE or text editor, create a new file named `hello.py`, and

```
1      print("Hello, World!")
```

This snippet of code does the following:

- It takes the argument, a string of characters that make up the word `Hello, World!`
- It passes that string into the built-in Python function `print`.

FIXME explanation of console and running the file

1.3 Datatypes and Variables

There are many different datatypes in Python, and all of them can be passed to `print` function. Here's the main few:

Strings Strings are just a sequence of characters. Anything closed between quotes gets included in the string, such as "This is the Python Chapter for the Kontinua Sequence!!%& × ()-+=1234567890 &"

Integers Integers are any whole number, positive negative or 0. Theoretically, integer values can go on for infinity, but it is important to understand they take up space in computer memory.

Floats Floats, or floating point values, are numbers that include decimal places. Division can be done between integers and floats but may result in a different value.

Booleans Booleans only two values: `True` or `False`. The output of conditional statements (such as if's or while's) are booleans. They only answer Yes and No questions, and are important on deciding between two possible paths in code.

Sequence Types Sequences are consecutive lists or pairs of other data types can be represented in various froms: `list`, `set`, `tuple`, and dict.

Let's create some of these in our code:

```
1      x = 5
2      y = 5.6
3      z = True
4      a = [x, y, z]
```

A *variable* is a container for information, usually containing one of the datatypes estab-

lished above. In Python, variables are assigned using the assignment operator =, with the name of the variable on the left side of the operator and the value it is assigned to on the right side.

In our code above, `x` is a *variable* containing the number 5, an Integer. `y` contains the float 5.6, while `z` is a boolean with the value `True`. We then created a list containing the variables `x`, `y`, and `z`.

A unique feature of Python is that variables can change their datatypes even after assignment. We can see the datatype of an object using the `type()`

```
1 x = 10
2 print(x)
3 print(type(x))
4
5 x = "Hello!"
6 print(x)
7 print(type(x))
```

Output:

```
10
<class 'int'>
Hello!
<class 'str'>
```

Notice that, because of that feature, the datatype of a variable is not defined beforehand. This is a core difference between Python and other programming languages like Java or C++, which we will explore in the future.

FIXME scientific notation floats

1.4 Arithmetic Operations

Math operations in Python are very similar to standard math operations.

```
1      a = 100
2      b = 3
3      c = 100.0
4      d = -100
5      print(a + b) # addition
6      print(a - b) # subtraction
7      print(a * b) # multiplication
8      print(a / b) # division (results in a float)
9      print(a % b) # modulus (remainder) operator
10     print(a ** b) # exponentiation
11     print(a // b) # floor division
12     print(c // b) # floor division as a float
```

```
13      print(d // b) # negative flooring goes further DOWN
```

Output:

```
103
97
300
33.333333333333336
1
1000000
33
33.0
34
```

Note that division by `0` is not a valid operation, just as in algebra. This will cause an `ZeroDivisionError`. FIXME -= and += FIXME expand, exercises

1.5 Conditionals, Loops, and Match-Case

1.6 Lists

1.7 Strings and Casting

We talked about creating strings, any set of characters between quotes. These can either be single quotes `' '` or double quotes `" "`.

Multiline strings must be surrounded by three quotations on each side of the text sequence.

```
1 a = """Lorem ipsum dolor sit amet,
2 consectetur adipiscing elit,
3 sed do eiusmod tempor incididunt
4 ut labore et dolore magna aliqua."""
```

1.8 Errors

Even the most experienced programmers can make mistakes. If the code seems to run into issues, your code may crash or output an *error*. Python reports errors by raising a halt in the console output, which include a message explaining the problem and the exact line where it occurred. By learning how to read and interpret these messages, you'll be able to *debug*, or fix issues, in your programs more efficiently and write more reliable code.

You will encounter two main types of errors in your code: **Syntax Errors** and **Runtime Errors/Exceptions**.

- Syntax errors occur when your the formatting, or *syntax*. These errors can usually be found before your code is compiled.
- Runtime Errors, or *Exceptions*, occur when operations in your code cause an invalid calculation, or attempt to do an invalid action. These can range anywhere from basic misnamed variables to imported libraries crashing from incorrect data.

1.8.1 Syntax Errors

What do you notice is immediately wrong with this code?

```
1     x = "Welcome Home"
2     print(type(x)
```

If we run it, we get the output:

```
Traceback (most recent call last):
  File "<string>", line 1, in <module>
  File "/usr/lib/python3.12/py_compile.py", line 150, in compile
    raise py_exc
py_compile.PyCompileError:   File "./prog.py", line 2
    print(type(x)
         ^
SyntaxError: '(' was never closed
```

We notice that every opening parentheses, bracket, or brace must have a closing supplement. Without out, we run into Syntax Errors, letting us know our format is off.

Another type of Syntax Error can be found in incorrect indentation. Take for example the following code:

```
1 if True:
2 print("Hello!")
```

Output:

```
Traceback (most recent call last):
  File "<string>", line 1, in <module>
  File "/usr/lib/python3.12/py_compile.py", line 150, in compile
    raise py_exc
py_compile.PyCompileError: Sorry: IndentationError: expected an indented bloc
```

Here, there is no indentation (usually obtained by pressing the `Tab` key on your keyboard) for lines under the conditional statement. This causes a Syntax Error to be raised.

1.8.2 Exceptions

Let's say I try to run the following code:

```
1      print(x)
2      x = "hello"
```

You may see the following output:

```
Traceback (most recent call last):
    File "./program.py", line 1, in <module>
NameError: name 'x' is not defined
```

You have encounted a *NameError*, because `x` was not assigned before it was attempted to be printed. This brings up an important note on Python code: **code is executed line-by-line, sequentially**. So even if you define `x` after you try and print it, Python will not understand what you are trying to print. Let's look at another example:

FIXME

1.8.3 Try-Except

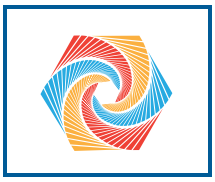
1.9 Libraries

1.10 An Input Calculator

1.11 Matplotlib

Appendix A

Answers to Exercises



Index