Course

Luxformel

Contents

1 The Python Programming Language

1.1 Introduction

Python is a high-level, general-purpose programming language. Its design philosophy emphasizes code readability with the use of significant indentation. Python was first developed by **Guido van Rossum**, a Dutch programmer, who invented Python to make programming easier.

Today Python is used across many disciplines such as *Machine Learning*, high level *Mathematics*, *Data Science* and *Data Analysis*.

1.2 Variables

Variables in Python allow us to store data, that we might later on use in the program.

1.2.1 Data Types

To store different data such as numbers, text, etc we use whats called *data types* these allow us to specify to the program which kind of data we meant.

There are many data types some of them include:

Name of Datatype	Type of dat	Example
String	Text	Storing a name in the code
Integer	Whole number	Storing a number of building in the code
Float	Decimal Number	Storing a Measurement in the code
Boolean	True or False value	Storing if a condition is met or not

> **Important**: Datatypes are not used for storing information on the computer but just to store them while the code is being executed.

1.2.1.1 Examples Different types of data being declared as variables, such as: Strings, Integers, Floats and Booleans respectively.

```
user_name = "Luxformel"
user_age = 25
height = 1.81
is_logged_in = True
```

1.2.1.2 More to Know : Naming in Programming

In Python, you can name variables almost anything, but how you format those names matters for readability.

Style and name	Format	Used In…
snake_case	All lowercase, underscores	**Python** (Standard for variables)
camelCase	First word lower, next words Capital	JavaScript, Java
PascalCase	Every word capitalized	Python (Classes), C#
kebab-case	Dashes between words	CSS, URL slugs

1.3 Typecasting

Typecasting means converting one *data type* into another.

1.3.1 Common Typecasting Use Case

A problem might arise in the following scenario:

```
age = "24"
```

Here we have declared a variable and assigned it to the *text* value or `String` of `24` and not the `integer` or `float` value of the actual number we might be trying to use. A `String` can not be used to calculate and doesn't have any of these possibilities of manipulation as the number values would have. This problem is fixed by what is know in programming as *Typecasting*.

1.3.1.1 Example

We *typecast* the string value of 24 to the integer value of 24.

```
age = int("24")
```

Now the value of age is the same as:

```
age = 24
```

Thus avoiding the problem of not being able to calculate using the number 24.

1.3.2 Typecasting Functions

These are only some available typecasting functions in python.

Data Type	Function	Example
String	`str()`	`name = str(42)` → `"42"`
Integer	`int()`	`age = int("24")` → `24`
Float	`float()`	`height = float("5.7")` → `5.7`

Data Type	Function	Example
Boolean	`bool()`	`is_active = bool("True")` → `True`

1.4 Print Function and f-Strings

The `print()` function sends information to the console. To make our output readable, we use **f-strings** (Formatted Strings).

F-strings allow you to "inject" variables directly into a sentence by placing an `f` before the quotes and using curly braces `{}`.

```
score = 100
# The modern, readable way:
print(f"Your final score is {score} points!")
```

1.5 Input

When first starting to use python and learning to program simple programs usually run not as we know it as standalone apps, but rather in the `terminal`.

1.5.1 Definition of the Terminal

The terminal is a text-based interface used to run commands, execute Python scripts, and manage your development environment.

> **Note**: Sometimes the terms console and terminal are used interchangeably, they mean almost the same thing.

1.5.2 Input Function

Simple programs can be written in python using the terminal. To however receive information the user of our program can give the program a sort of input is necessary, this is what the `input()` function solves.

The `input()` function however always *returns* a string. Therefore when using the function we sometimes use *type casting* to use the correct *data type*.

1.5.2.1 Example

A simple program which prints the users age:

```
print("Please enter your age:")
age = int(input())
print(f"Your age is: {age}")
```

> **Notice**: We have used the previously learned functions, and programming techniques. This is not uncommon for programming, usually ever single aspect learned should be remembered as it will usually be important later on.

1.6 Math in Python

Python can perform calculations instantly. Most operators look like standard math symbols:

Operator	Action	Example	Result
`+`	Addition	`10 + 5`	`15`
`-`	Subtraction	`10 - 5`	`5`
`*`	Multiplication	`10 * 5`	`50`
`/`	Division	`10 / 4`	`2.5`
`//`	Floor Division	`10 // 4`	`2` (removes decimal)
`%`	Modulo	`10 % 3`	`1` (the remainder)

1.7 Conditionals

Conditionals allow a program to make decisions. They let your code run different blocks of code depending on whether a condition is true or false.

In Python, conditionals are mainly written using the keywords: `if`, `elif` and `else`.

1.7.1 Comparison Operators

To create conditions, we often compare values using comparison operators:

Operator	Meaning	Example	Result
`==`	Equal to	`5 == 5`	`True`
`!=`	Not equal to	`5 != 3`	`True`
`>`	Greater than	`10 > 3`	`True`
`<`	Less than	`3 < 10`	`True`
`>=`	Greater or equal	`5 >= 5`	`True`
`<=`	Less or equal	`4 <= 2`	`False`

> **Important:**
> `=` is used for assignment,
> `==` is used for comparison.

1.7.2 The `if` Statement

The simplest conditional uses `if`.

1.7.2.1 Example

This is simple code to *compare* a number to 10.

```
number = 50

if number > 10:
    print("Your number is bigger than 10!")
```

> **Notice:**
> Python uses indentation (spaces) instead of braces `{}`.
> Everything indented under `if` belongs to it.

You might notice this is not sufficient for expressing more complicated logic in a program, as usually even in language an **if** is followed by **else**.

1.7.3 The `if`, `else` Statement

The `if else` Statement allows us to expand our previous program to be a bit more complicated but more intuitive:

1.7.3.1 Example

This is simple code to *compare* a number to *another* number.

```
print("Please enter a number:")
number = int(input())

if number > 10:
    print("Your number is bigger than 10!")
else:
    print("Your number is smaller than 10!")
```

1.7.4 The `if`, `elif`, `else` Statement

To expand the `if` statement even more and add more possibilities than 1 or 2, we can use the `elif` statement. `elif` stands for *else if* this allows us to add more conditions.

1.7.4.1 Example

This programs allows us to see if a number is positive, negative or zero.

```
print("Please enter a number:")
number = int(input())
```

```
if number > 0:
    print("Your number is positive!")
elif number < 0:
    print("Your number is negative!")
else:
    print("Your number is 0!")
```

> **Note**: The `elif` statement can be repeated multiple times to add as many conditions as possible.

1.7.5 Nested Conditionals

A more powerful way of writing conditionals is sometimes to write `if` statements inside other `if` statements. This is called *nesting*.

1.7.5.1 Example

Checking if a user is logged in and also a user.

```
is_logged_in = True
is_admin = False

if is_logged_in:
    if is_admin:
        print("Welcome, admin!")
    else:
        print("Welcome, user!")
else:
    print("Please log in.")
```

1.8 Logical Operators

With the last example on the section on *Conditionals* you may have noticed that nesting the conditionals, required us to use the word *and* this idea of two conditionals being checked by a single if statement - as supposed to nesting them - is the idea behind logical operators.

Logical operators allow you to combine multiple conditions or invert conditions in Python. They are especially useful when writing complex if statements.

Operator	Meaning	Example	Result
and	True if both conditions are true	True and False	False
or	True if at least one condition is true	True or False	True

Operator	Meaning	Example	Result
not	Reverses the value of the condition	not True	False

1.8.1 `and` Operator

The `and` operator specifies that both conditions at the same time must be `True` for the condition to be `True` as well.

On the contrary if only one of them is `False` the entire condition will also be `False`.

1.8.1.1 Example

This code uses the `and` operator to grant or deny access to an event to a user.

```
age = 25
has_ticket = True

if age >= 18 and has_ticket:
    print("You may enter the event.")
else:
    print("Access denied.")
```