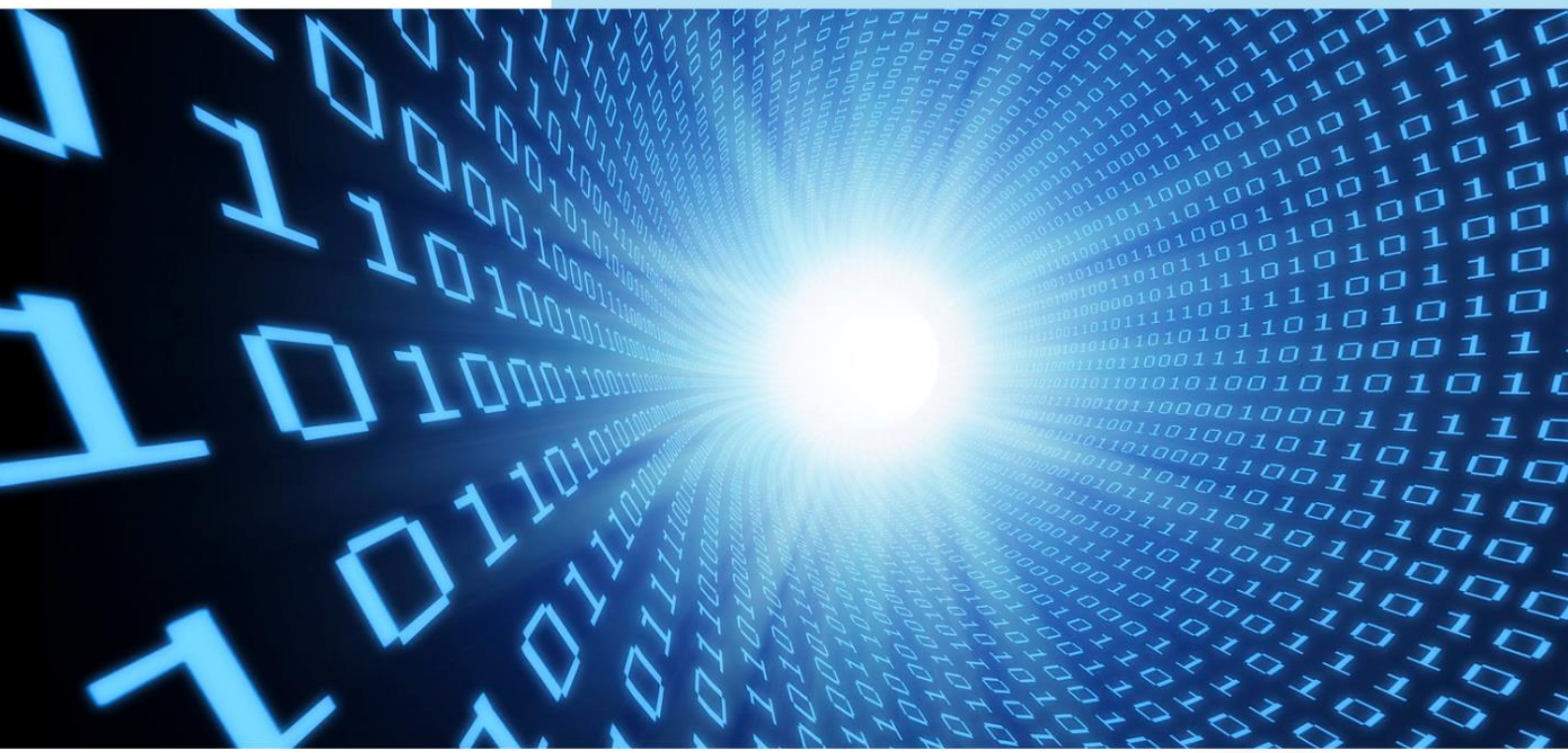
LPWR ACADEMY



PYTHON FROM ZERO

LEARN PYTHON
A BEGINNERS GUIDE TO PYTHON

A 70+ pages learning journey to master Python from scratch.

LEARNPYTHONWITHRUNE.ORG

The Simple, Intuitive, and Intended Way

Did you know that most people don't understand what programming is?

Because of this, many people struggle to learn about programming and spend too much time learning insignificant aspects of it.

A senior programmer knows that code is written once, but read hundreds of times.

How does that change their approach to programming?

Well, a programmer knows that it's about creating the most efficient, unintuitive, and complex code constructions. They know that **code** is about **readability**.

Code should be intuitive and straightforward to understand.

Why Python?

Python is ideal for programmers because the syntax is minimal and constructed with **readability** as a core focus point. In addition, **Python** can be used for simple scripts and full-scale object-oriented programming.

Simplicity is the key to **Python**'s success and popularity.

Most people do not understand this aspect of **Python** and, more importantly, programming.

This means that many courses teach **Python** and programming incorrectly, making it too complex, covering too many topics, and teaching edge-case optimizations.

Senior programmers make their **code easy to understand and read**.

Why do most teachers not understand this?

Well, because they teach coding and have little to no experience with production code. Their code is written once and then never seen again. They have an academic and not practical approach to coding.

Why does my teaching stand apart from most teachers?

Hello! I am Rune from Denmark, and I have been programming professionally for over 15 years. In addition, I have had a passion for teaching since I began studying at university for my Ph.D.

Copyright © 2023 learnpythonwithrune.org

Learn Python

This **eBook** covers everything you need to know to get started with Python.

What will this eBook teach you?

This **eBook** will cover the following topics:

- Understanding and using variables
- Using lists and dictionaries:
- Program flow with if-statements and loops
- How to create functions
- Using randomness in a program
- Creating simple games
- Reading and processing CSV files
- Object-Oriented Programming (OOP)
- Visualization of data
- Creating projects with NumPy and Pandas
- Creating 17 projects, including a Machine Learning model from scratch
- Recursive functions
- List comprehension

How to get the most out of this eBook

This **eBook** is accompanied by **video lessons** and **Jupyter** Notebooks available from my **GitHub repository**.

Each chapter has a corresponding **video tutorial,** which covers the new concepts, introduces a project, and gives a solution for the project (**see course page**).

Using the **Notebooks** along with the **eBook** will improve your learning journey.

There are two ways to move forward.

- Go to my **GitHub repository.** Then, download all Notebooks and work with them in your **Jupyter Notebook** environment (**you can download all Notebooks as a Zip file**).
- Use the links to **Colab** given in the **GitHub description** and work with the **Notebooks** interactively in **Colab** (no need to install setup).

Installing your setup (recommended)

Anaconda comes with **Jupyter Notebook** and **Python**, which is all you need. Install **Anaconda Individual Edition** and Launch **Jupyter Notebook**. See the **documentation for troubleshooting**.

Copyright © 2023 learnpythonwithrune.org

Table of Content

Copyright © 2023 learnpythonwithrune.org

Copyright © 2023 learnpythonwithrune.org

Copyright © 2023 learnpythonwithrune.org

Copyright © 2023 learnpythonwithrune.org

Copyright © 2023 learnpythonwithrune.org

00 – Hello World

In this chapter, we will get acquainted with **Python** and **Jupyter Notebook**.

Learning Objectives

- Get acquainted with the **Jupyter Notebook**
- How to execute the cells (i.e., the Python code)
- Get input from user
- Print strings
- Simple string methods

A brief introduction to the Jupyter Notebook

If you are new to **Jupyter Notebook,** this section is for you. Otherwise, you can skip to the next section, where we will learn about the coding aspect.

What is Jupyter Notebook?

Jupyter Notebook is an open-source web application that allows you to create and share documents that contain live code, equations, visualizations, and narrative text.

Essentially, Jupyter Notebook:

- Works in your browser;
- Allows you to write **Python** code in it;
- Executes the **Python** code for you and shows the results;
- Allows you to write text in it; and
- Creates visualizations with charts.

Jupyter Notebooks are popular among beginners and **data scientists** because:

- It is easy to learn programming in a **Notebook**;
- The needs of a **data scientist** are well served in a **Notebook**; and
- **Notebooks** are good for exploring data and programming easily.

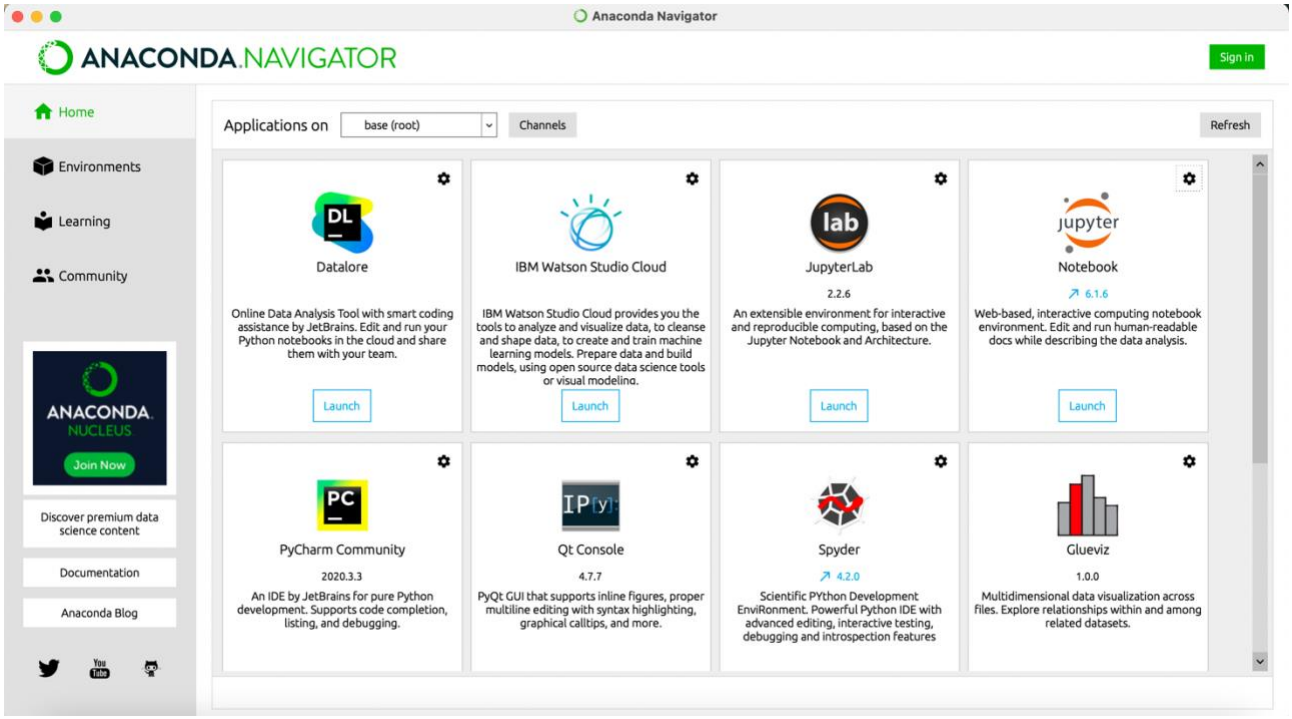
Jupyter Notebook seems to be a perfect match for starters.

How to open the Jupyter Notebooks from this eBook

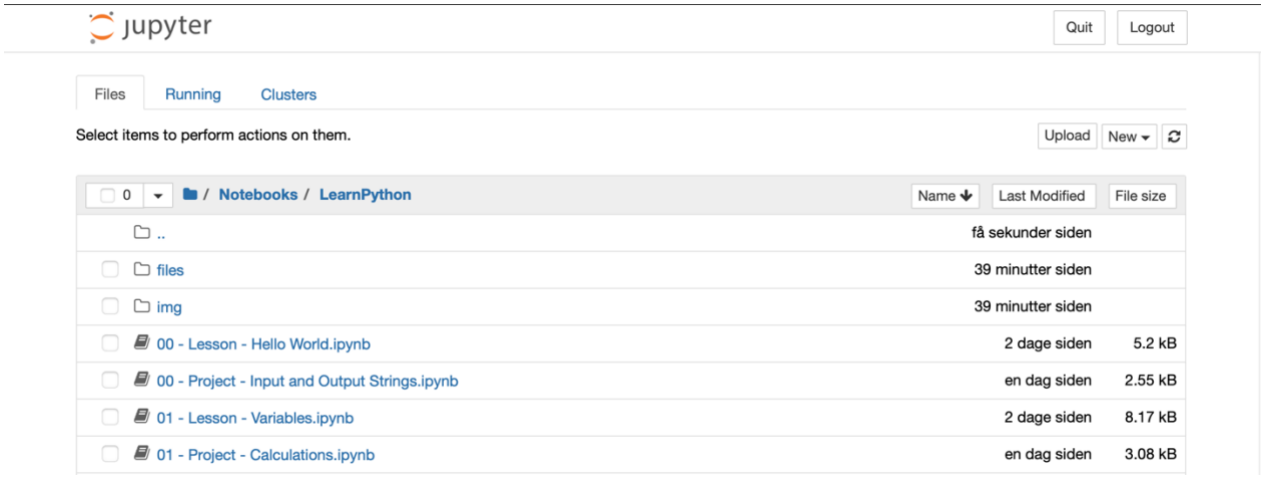
To launch Jupyter Notebooks:

1. Install **Anaconda Individual Edition**;
2. **Download the zip file from the GitHub**;
3. Unzip the content of the zip file in a location you can find;
4. Launch **Anaconda Navigator** from your main menu (Launchpad or Windows menu); and
5. Inside **Anaconda Navigator** launch **Jupyter Notebook**.

Copyright © 2023 learnpythonwithrune.org



These instructions should bring you into your main browser with the **Jupyter Notebook Dashboard** (see below).



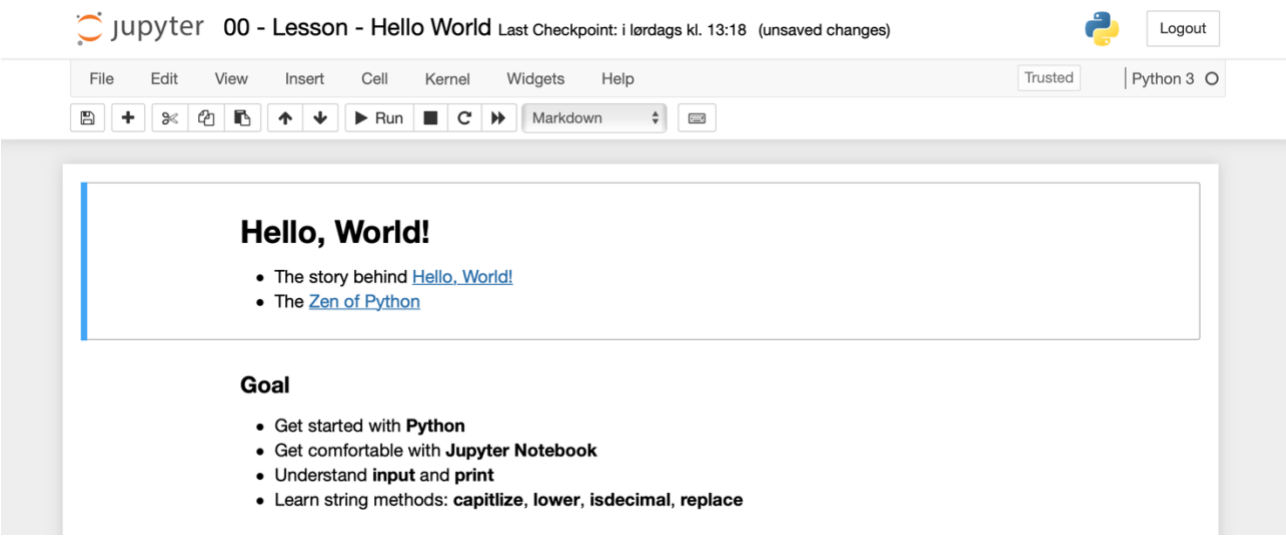
- Navigate to the extracted files from the downloaded zip file.
- Click on the first **Notebook** (00 – Lesson – Hello World.ipynb)

This will open the **Notebook** in a new browser tab.

How to execute the code in the Jupyter Notebook

A **Notebook** is divided into cells.

Copyright © 2023 learnpythonwithrune.org



A cell can contain text like the two examples above. The blue box indicates the first cell.

The cell containing the code can be executed.

```
In [ ]: print("Hello, World!")
```

The above cell contains one line of code: a **print** statement.

A cell has been executed if it has a number. The one above has not been executed: **In [ ]:**

An executed cell would look like this.

```
In [1]: print("Hello, World!")
        Hello, World!
```

Executing a cell can be done in various ways. The usual practice is to mark the cell, which can be done with a mouse or keys and pressing enter.

```
In [ ]: print("Hello, World!")
```

Then press **Shift + Enter** to execute it.

Alternatively, you can press **Run** with your mouse.

The cell will execute and finishes with a number: **In [1]:**.

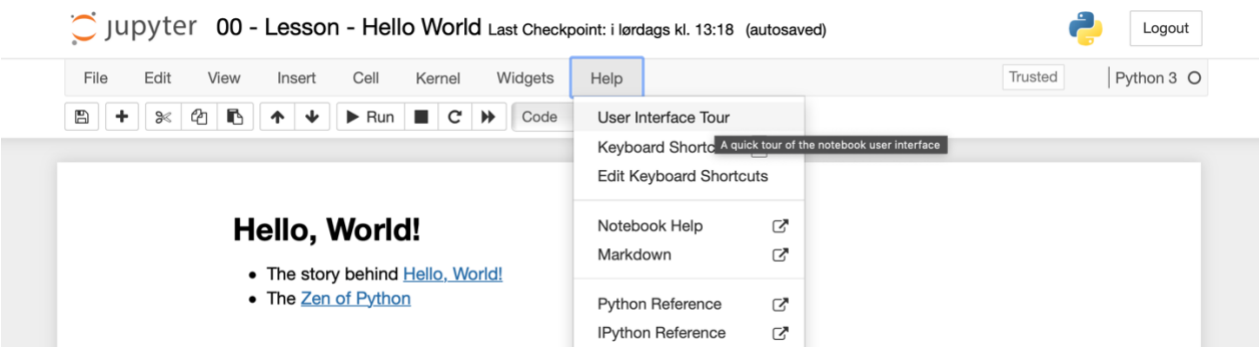
```
In [1]: print("Hello, World!")
        Hello, World!
```

To edit a cell, just mark it and press **enter.** Then, you enter edit-mode.

This is all you need to know to follow along with the **eBook**.

If you would like to learn more about the specific elements within the Notebook Editor, you can go through the user interface tour by selecting **Help** in the menu bar, then selecting **User Interface Tour**.

Copyright © 2023 learnpythonwithrune.org



Python print statement

Print statement.

```
In [1]: print("Hello, World!")
        Hello, World!
```

Print a string or variable to the console. In **Jupyter Notebook,** the print statement will output below the executed cell.

A print statement can take any number of arguments and output them separated with a space. This is a great way to print variables of different types to avoid mistakes with types.

```
In [2]: print("Hey", 10, 2.3)
        Hey 10 2.3
```

Python input statement

You can prompt the user to input a string and assign it to a variable.

```
In [ ]: s = input()
```

When executed, you (the user) will be prompted.

```
In [*]: s = input()
```

And you can type a message ending it by pressing enter.

```
In [3]: s = input()
        Rune
```

The cell will be executed and assigned **s** to the string, *"Rune" (if you entered Rune).*

You can prompt a question with the input statement.

```
In [*]: s = input("What is your name?")
        What is your name?
```


Copyright © 2023 learnpythonwithrune.org

A few string methods

A great thing to master is a few string methods, making string manipulation easy for you.

An example is to capitalize a string.

```
In [5]: "rune".capitalize()
Out[5]: 'Rune'
```

Notice that all the string methods can be called directly on strings (as above) or a variable containing a string (as below). Also, you can store the result in a new variable.

```
In [12]: s = "rune"
         t = s.capitalize()
         t
Out[12]: 'Rune'
```

The above shows the output of the variable on the cell's last line.

We will learn more about variables in the next lesson. Here we will just get familiar with some basics.

Convert to uppercase.

```
In [6]: "Rune".upper()
Out[6]: 'RUNE'
```

Lowercase.

```
In [7]: "Rune".lower()
Out[7]: 'rune'
```

Check if the string is a decimal number.

```
In [8]: "223".isdecimal()
Out[8]: True
```

And replace a substring with another string.

```
In [13]: name = "Rune"
         name.replace('R', 'S')
Out[13]: 'Sune'
```

How to avoid the new line in a print-statement

You can print a string without a new line as follows.

```
In [16]: print("No new line", end='')
         print("New line")
         No new lineNew line
```

Where the **end=''** replaces the end of the line with an empty string, thereby overwriting the new line.

Copyright © 2023 learnpythonwithrune.org

01 – Variables and Types

In this chapter, we will learn about variables in Python and the basic types.

Learning Objectives

- What is a variable
- The basic types: **integers**, **floats**, **strings**, and **Booleans**
- An overview of some essential math functions

Variables and declarations

A variable is a reserved memory location. In Python, a variable gives a program the data to the computer for processing.

Without variables, there is no data to process.

Declaration of variables

```
In [1]: a = 2
        b = 2.4
        c = "This is a string"
```

The above shows how to declare variables.

As you might conclude, the variables have types. We will cover the basic types in a moment. First, we need to learn about the naming convention of variables.

Python Variable Naming Convention

A variable name in Python should fulfill the following:

- Be in lowercase font, or
- Have separate words with an underscore

A variable name in Python can be:

- A single lowercase letter,
- A lowercase word, or
- a lowercase word separated by an underscore.

Examples of variable names:

- **x**
- **i**
- **cars**
- **my_cars**

Note that other programming languages can have other naming conventions.

Copyright © 2023 learnpythonwithrune.org

Integers

An integer is a whole number and not a fraction.

Examples of integers:

- -23
- -5
- 0
- 1
- 78
- 1,980,350

Examples of non-integers:

- 3.14
- 0.001
- 9,999.999

In Python, you can use the **type** built-in function to get the type of variable or value.

```
In [2]: type(3)
Out[2]: int
```

Declarations of integers are made simply by assigning an integer to a variable.

```
In [3]: a = 2

In [4]: type(a)
Out[4]: int
```

You can do simple arithmetic with variables and keep the type.

```
In [5]: a = 1
        b = 2
        c = a + b
        type(c)
Out[5]: int
```

Floats

A **float** is a number with digits on both sides of a decimal point. This contrasts the integer data type, which houses an integer or whole number.

Examples of floats:

- 3.13
- 9.99
- 0.0001

Note that the **float** data type in **Python** can represent a whole number.

- 1.0

Copyright © 2023 learnpythonwithrune.org

- 3.0

Declaration of a float.

```
In [6]: a = 3.13
        type(a)
Out[6]: float
```

Simple arithmetic with floats.

```
In [7]: a = 1.0
        b = 2.0
        c = a + b
        type(c)
Out[7]: float
```

Finally, notice that the arithmetic of the integers (int) and floats will convert the result to float.

```
In [8]: a = 1
        b = 1.0
        c = a + b
        type(c)
Out[8]: float
```

A Few Math Functions

It is always useful to master a few math functions.

The absolute value of a value or variable.

```
In [9]: abs(-2)
Out[9]: 2
```

Variables.

```
In [10]: x = -4
         abs(x)
Out[10]: 4
```

The power function.

```
In [11]: pow(2, 3)
Out[11]: 8
```

Rounding is often handy when you output numbers.

```
In [13]: round(2.35, 1)
Out[13]: 2.4
```

The round function also works directly on variables.

Strings

A string in Python is a sequence of characters.

Copyright © 2023 learnpythonwithrune.org

Examples:

- "I am a string!"
- “I am another string.”

We have already used strings. The above shows that you can use double or single quotes.

```
In [14]: s = "This is a string"
```

A great tool to master is formatted strings. A formatted string can insert another variable into the string. See this example:

```
In [15]: a = 1
         print(f"This is {a}")
         This is 1
```

Here, it inserts the value of variable “**a”** into the string.

How did that happen? First, you need to use a **f** in front of the string. Then, you use curly brackets **{}** for the variables.

You can insert as many variables as you want into the string.

Boolean

Boolean represents one of two values: **True** or **False**. Boolean is often used to structure the flow of a program.

Here, we will learn about how **Boolean** expressions are considered. Later, we will combine it with **if**-statements to create a flow in a program based on that.

```
In [16]: b = True
         b
Out[16]: True
```

Note: It is case-sensitive. If you write **true** (instead of **True**), it will not work.

```
In [17]: b = False
         b
Out[17]: False
```

Yes, those are the two values. I agree, that doesn’t look very useful.

The power comes when you evaluate expressions to a **Boolean** value.

```
In [18]: b = 10 < 12
         b
Out[18]: True
```

It looks confusing. But we evaluate the expression **10 < 12** and assign it to **b**. This is true, obviously.

Later, when we combine this with **if**-statements, we will be able to create flows, and it will all make sense.

For now, if you like, you can evaluate expressions by using the built-in Python function **bool**.

Copyright © 2023 learnpythonwithrune.org

```
In [19]: bool(2 > 4)
Out[19]: False
```

This will teach a lot of things. You can play around with strings, integers, floats, and more to learn about it.

Copyright © 2023 learnpythonwithrune.org

02 – Type Conversion

In this chapter, we will learn about types and conversion between types.

Learning Objectives

- Identify problems with types.
- Learn about type conversion with **int(), float(),** and **str()**.

A Problem with Types

Let me demonstrate.

Imagine the following code.

```
In [1]: name = input("What is your name?")
        print(f"Hi, {name}!")
        birth_year = input("What is your birth year?")
        print(f"You are born in {birth_year}")
```

```
What is your name?Rune
Hi, Rune!
What is your birth year?1991
You are born in 1991
```

All goes as expected. Notice the formatted string we use.

Now, try this code.

```
In [ ]: name = input("What is your name?")
        print(f"Hi, {name}!")
        birth_year = input("What is your birth year?")
        print(f"You are {2021 - birth_year} years old")
```

We changed it a bit and want to calculate how many years you are (assuming we were in 2021).

Let's try.

```
In [2]: name = input("What is your name?")
        print(f"Hi, {name}!")
        birth_year = input("What is your birth year?")
        print(f"You are {2021 - birth_year} years old")
```

```
What is your name?Rune
Hi, Rune!
What is your birth year?1991

---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-2-9d25aa9c1fe3> in <module>
      2 print(f"Hi, {name}!")
      3 birth_year = input("What is your birth year?")
----> 4 print(f"You are {2021 - birth_year} years old")

TypeError: unsupported operand type(s) for -: 'int' and 'str'
```

Somethings wrong. A **TypeError**.

When you see this the first time, it can be tricky to figure out what is wrong.

To narrow it down, it does not like the **2021 – birth_year**.

Copyright © 2023 learnpythonwithrune.org

We know that **2021** is an integer (**int**), but it turns out that **birth_year** is a string (**str**).

```
In [3]: type(birth_year)
Out[3]: str
```

Unfortunately, we cannot subtract an integer and string.

So, now what do we do?

Conversions.

Convert String to Integers

Luckily, we can easily convert a string to an integer with the **int()** built-in **Python** function.

```
In [4]: birth_year = int(birth_year)
        type(birth_year)
Out[4]: int
```

Now, let's try the following.

```
In [5]: name = input("What is your name?")
        print(f"Hi, {name}!")
        birth_year = input("What is your birth year?")
        birth_year = int(birth_year)
        print(f"You are {2021 - birth_year} years old")

        What is your name?Rune
        Hi, Rune!
        What is your birth year?1991
        You are 30 years old
```

I love it. I am 30 years old. Wait a minute. I wasn't born in 1991.

Nevertheless, the above code works now.

Convert String to Float

Similarly, you can convert a string to float.

```
In [6]: a = "3.14"
        pi = float(a)
        type(pi)
Out[6]: float
```

That is nice.

Convert Integer to String

It is also possible to convert an integer to a string.

Copyright © 2023 learnpythonwithrune.org

```
In [7]: a = 2
        b = "This is a value "
        c = b + str(a)
        c

Out[7]: 'This is a value 2'
```

Notice that strings added together are concatenated.

Copyright © 2023 learnpythonwithrune.org

03 – Program Flow

Program flow is a general term that describes how your lines of code are executed.

Learning Objectives

- **Boolean** expressions
- Understand **if**-statements to create a **program flow**
- How **Boolean** expressions are used in **if**-statements
- To connect a **program flow** with a decision tree

Boolean Expressions

The **Boolean expressions** are the ones we touched upon earlier, where we learned about the data type **Boolean**.

The **Boolean expressions** are the following:

- Equals: a == b
- Not Equals: a != b
- Less than: a < b
- Less than or equal to: a <= b
- Greater than: a > b
- Greater than or equal to: a >= b

If-statement

An **if**-statement is as follows.

```
In [1]: a = 10
        b = 12
        if a < b:
            print("a is less than b")

        a is less than b
```

The **if**-statement (**if a < b:**) evaluates the **Boolean expression** (**a < b**), if it is true, it will execute the following indented code (here: **print("a is less than b")**), if not, then it will not continue after the indented code.

Just to demonstrate the difference:

```
In [2]: a = 10
        b = 12
        if a > b:
            print("a is greater than b")
```

The cell above does not output anything, as **a** is not greater than **b**.

Copyright © 2023 learnpythonwithrune.org

If-else-statement

We can add an **else**-statement as follows.

```
In [3]: a = 10
        b = 12
        if a < b:
            print("a is less than b")
        else:
            print("a is not less than b")
```

```
a is less than b
```

The **else**-statement will execute the following indented code if the **if**-statement is false. The above is **True**, hence the else-statement code will not be executed.

To demonstrate, we can change the value of **a**.

```
In [4]: a = 15
        b = 12
        if a < b:
            print("a is less than b")
        else:
            print("a is not less than b")
```

```
a is not less than b
```

If-elif-else-statement

You can create more branches in your code as follows.

```
In [5]: a = 10
        b = 10
        if a < b:
            print("a is less than b")
        elif a == b:
            print("a equals b")
        else:
            print("a is not less than b")
```

```
a equals b
```

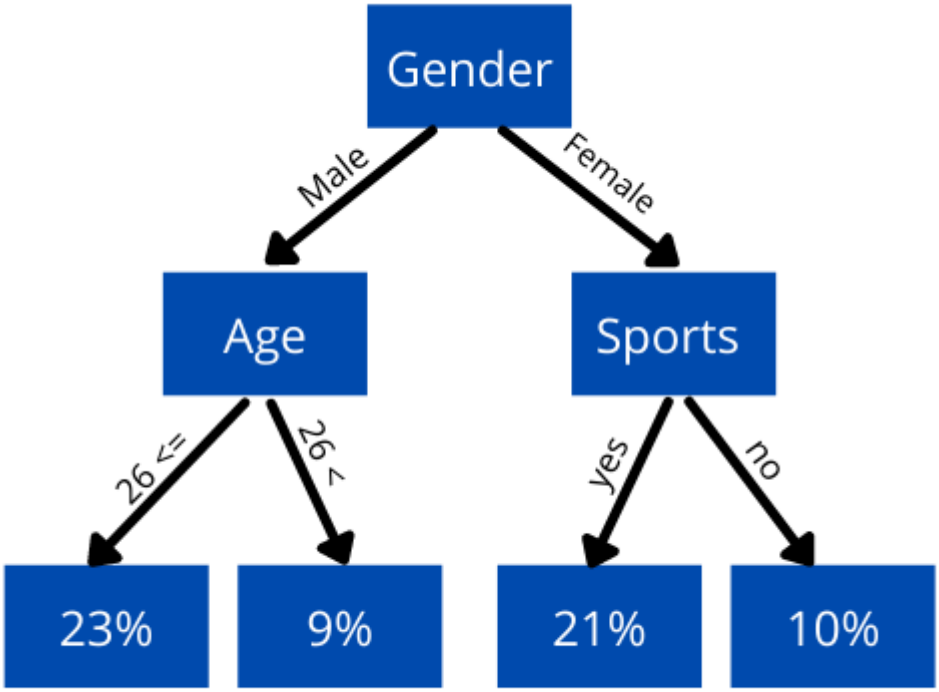
This will execute the first **True if** or **elif**-statements indented code. If none are True, the **else**-part will be executed.

Other flows

Later we will learn about **for**-loops and **while**-loops, which is a way to create iterative flow in your programs.

For now, the above syntax will help you create flows of decision trees.

Copyright © 2023 learnpythonwithrune.org



In this project, we will implement the above decision tree.

Copyright © 2023 learnpythonwithrune.org

04 – Random

Getting random into our programs enables us to create many fun things, like simple games.

Learning Objectives

- Learn about libraries
- Import the random library
- Use random in our program

What is a library?

A **Python** library is a reusable chunk of code that you may want to include in your programs.

What can a library look like?

It can:

- Be functionalities to use in your program (like getting randomness);
- Consist of data structures, like **NumPy** arrays and **Pandas DataFrames**, which we will learn about later in this course; or
- Help create visualizations, like **Matplotlib**, which is also covered later in this course.

You can also think of libraries as building blocks to help you craft your programs.

Get started with Random

The best way to learn about libraries is to get started and see how it works.

The first thing you need is to **import** it to tell that you need access to it.

```
In [1]: import random
```

Now, you can access modules and functions from the **random** library.

How do you know what you can do?

The simple answer is documentation. Starting as a programmer, documentation can be a bit frightening to look at. Everything is documented, so where do we begin?

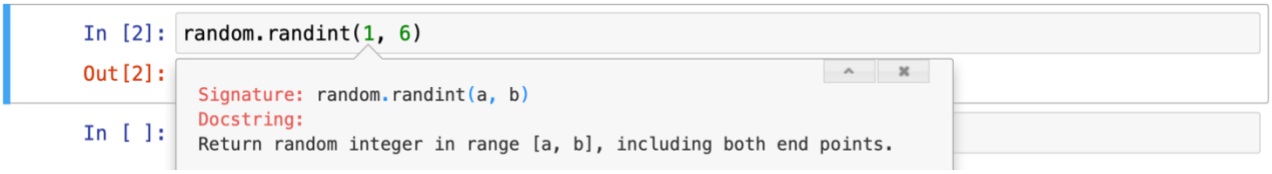
If interested, see the **random docs here**.

I must admit, the random docs are not that bad, but let's ignore the docs for now and get started with what we need.

Return a Random Integer

Often, we need to sample a random integer. Say we want to sample the roll of a die.

Copyright © 2023 learnpythonwithrune.org



First, you can get the doc string from a method or function in **Jupyter** Notebook. As you see here, the above **random.randint(1, 6)** returns a random integer in the range 1 to 6, including both.

Press **shift+tab** anywhere on the method or function you want the doc string from to get the doc string.

In this case, the **random.randint(1, 6)** returned as follows.

```
In [2]: random.randint(1, 6)
Out[2]: 3
```

If we rerun, we might get another value.

We can save the result in a variable as expected.

```
In [3]: a = random.randint(1, 6)
        a
Out[3]: 2
```

Return a Random Float

To get a float can be done with a uniform distribution.

```
In [4]: random.uniform(0, 1)
Out[4]: 0.5686488494583418
```

This means that all float numbers between 0 and 1 are equally like to be chosen.

If you are into statistics, you can also make a normal (or Gauss) distribution.

```
In [5]: random.gauss(0, 1)
Out[5]: -1.1291709893952773
```

Other useful functions

We will later use other functions from **random**, like getting a random element from a **list** or shuffling a **list**. As we have not covered a **list** yet, we will not introduce it in this chapter.

Copyright © 2023 learnpythonwithrune.org

05 – Lists

We will learn to work with Python **lists**.

Learning Objectives

- What is a **Python list**?
- How to use **lists**
- Get a random element from a **list**
- How to randomize a string – using a **list** in the process

What is a Python List?

Lists are used to store multiple items in a single variable.

A **Python list** can be used to store any value, objects, and even **lists**. Yes, a **list** can store **lists**.

Python lists are one of the built-in data structures in **Python**. To be honest, **Python lists** were one of the reasons I fell in love with **Python**. I know it sounds crazy, but I was working with the **C** programming language. Go figure how you would do something similar in **C**!

How to create a Python List

You can create a **list** by using square brackets **[]**, either an empty or one with elements as below.

```
In [1]: my_list = ["Apple", "Orange", "Banana"]
```

The list looks like this.

```
In [2]: my_list
Out[2]: ['Apple', 'Orange', 'Banana']
```

How to Access Elements in a List

You can access an element in a **list** by an index.

```
In [3]: my_list[0]
Out[3]: 'Apple'
```

Starting from 0 and forward.

```
In [4]: my_list[1]
Out[4]: 'Orange'
```

You can index backward with a negative index.

```
In [5]: my_list[-1]
Out[5]: 'Banana'
```


Copyright © 2023 learnpythonwithrune.org

Hence, index -1 gives the last element, and index -2 gives the second last, and so forth.

```
In [6]: my_list[-2]
Out[6]: 'Orange'
```

Adding and Removing Items from a List

You can add an element to the end of the **list** as follows.

```
In [7]: my_list.append("Apple")
```

```
In [8]: my_list
Out[8]: ['Apple', 'Orange', 'Banana', 'Apple']
```

This also shows that you can have duplicate elements in a **list**.

You can remove elements by using **pop**. By default, **pop** removes the last item.

```
In [9]: my_list.pop()
Out[9]: 'Apple'
```

And now the **list** looks like this.

```
In [10]: my_list
Out[10]: ['Apple', 'Orange', 'Banana']
```

Concatenate Lists

Let's create another **list**.

```
In [11]: another_list = ['Sweden', 'Norway', 'Denmark']
         another_list
Out[11]: ['Sweden', 'Norway', 'Denmark']
```

Concatenation is done by addition.

```
In [12]: new_list = my_list + another_list
         new_list
Out[12]: ['Apple', 'Orange', 'Banana', 'Sweden', 'Norway', 'Denmark']
```

Random Choice and Shuffle

Sometimes you need a random element from a **list**.

To do that, we need to **import random**.

```
In [13]: import random
```

Then we can get a random element as follows.

Copyright © 2023 learnpythonwithrune.org

```
In [14]: random.choice(new_list)
Out[14]: 'Norway'
```

If we re-run it, you most likely get another element.

```
In [15]: random.choice(new_list)
Out[15]: 'Apple'
```

Sometimes you want to **shuffle** the **list**.

```
In [16]: random.shuffle(new_list)

In [17]: new_list
Out[17]: ['Apple', 'Orange', 'Sweden', 'Banana', 'Denmark', 'Norway']
```

Useful String and List Methods

If you have a list of strings, you can concatenate them on a string by **join**.

```
In [18]: ' '.join(new_list)
Out[18]: 'Apple Orange Sweden Banana Denmark Norway'
```

This is magic.

Notice the space in the string (**' '.join(...)**). What **join** does, is concatenate each element in the **list**, putting the string it called on (here space) between each element.

Now, this is awesome. Check this.

```
In [18]: s = "awesome"
         random.sample(s, len(s))
Out[18]: ['s', 'm', 'e', 'a', 'w', 'o', 'e']
```

random.sample(s, len(s)) takes **len(s)** random unique samples from **s.**

Once more.

len(s) takes the length of the string **s.**

And **random.sample(s, len(s))** takes unique samples from **s.** As it takes **len(s)** unique samples, it means returning the characters of **s** in random order.

Wow.

Wait a minute.

```
In [20]: s = "awesome"
         ''.join(random.sample(s, len(s)))
Out[20]: 'aewomse'
```

If we call **join** on an empty string, we get **s** shuffled.

We will use that in our project for this chapter.

Learn Python with Rune

Copyright © 2023 learnpythonwithrune.org

Some Useful Functions for Lists with Numbers

Let's create a **list** of integers.

```
In [21]: my_numbers = [3, 7, 1, 9, 3, 6]
```

Then you can get the maximum number in the **list** as follows.

```
In [22]: max(my_numbers)
Out[22]: 9
```

The minimum element of the **list**.

```
In [23]: min(my_numbers)
Out[23]: 1
```

The sum of the **list**.

```
In [24]: sum(my_numbers)
Out[24]: 29
```

You get the length of the list by using **len( ).**

```
In [25]: len(my_numbers)
Out[25]: 6
```

And then, you can calculate the average as follows.

```
In [26]: sum(my_numbers)/len(my_numbers)
Out[26]: 4.833333333333333
```


Copyright © 2023 learnpythonwithrune.org

06 – Loops

In this chapter, we will learn how to use **loops** in our program flow.

Learning Objectives

- What is a **loop**?
- How to **loop** over **lists**
- How to **loop** over the characters in a string
- Cover both **for**- and **while**-**loops**

What is a loop?

A **loop** is a programming structure that repeats a sequence of instructions until a specific condition is met.

You can use **loops** to cycle through values, add sums of numbers, repeat functions, etc.

Your First Loop – Looping over a List

Consider this code and inspect the output.

```
In [1]: my_list = ['First', 'Second', 'Third']

        for item in my_list:
            print(item)
```

```
First
Second
Third
```

Makes sense?

The **for**-loop loops over the items in the list **my_list**, and for each item, it executes the indented code.

Looping over the Characters in a String

This can be done as follows.

```
In [2]: name = "Rune"

        for c in name:
            print(c)
```

```
R
u
n
e
```

Again, the same structure. The variable **c** is assigned to each character of the string **name**, and it executes the indented code for each one.

Copyright © 2023 learnpythonwithrune.org

Looping over a Range

Often, you need to loop over a range.

```
In [3]: for i in range(4):
            print(i)
```

```
0
1
2
3
```

Notice that we can use a combination of **len** and **range** to get a loop over a list.

```
In [4]: my_list = ['First', 'Second', 'Third']

        for i in range(len(my_list)):
            print(my_list[i])
```

```
First
Second
Third
```

Sometime, this syntax is more convenient, for example if you need the index of the element.

While loops

While-loops are amazing program flow when you have an unknown number iterations.

See the following example:

```
In [5]: value = 10

        while value > 0:
            value = int(input("Input value: "))
```

```
Input value: 5
Input value: 0
```

As you see, it can stop after the first iteration or never stop. It all depends on the result in the loop. This is different from **for**-loops.

Copyright © 2023 learnpythonwithrune.org

07 – Functions

In this chapter, we will learn how to use **functions** to structure and reuse code in our program.

Learning Objectives

- What is a **function**
- Why we use functions
- How to define **functions**
- **Functions** with arguments
- **Function** returning values

What is a Function?

A function is a block of reusable code. Some ways we use functions:

- To make code more **readable**
- To **reuse** the same code
- To **organize** code better
- To enables coding on a higher **abstraction**

Your First Function

The syntax of a function in **Python** is as follows.

- A function is defined by **def** in **Python**
- Followed by a unique name of your choice
- Then parenthesis with optional parameters (arguments)
- A colon ending the first line
- The following indented code is the function code
- The function can have return statements anywhere in the code

Let's start with a simple example:

```
In [1]: def my_func():
            print("Hello, World!")
```

Here we define the function **my_func(),** it has no arguments (we will get back to that later). The function simply prints **Hello, World!**

Notice that the cell is executed but did not output anything. That is because it is simply a definition.

To execute the code, we need to call the function.

Copyright © 2023 learnpythonwithrune.org

```
In [2]: my_func()
        Hello, World!
```

Any code block can be put into the function.

As you see, this can help you structure your code and not write the same code twice.

Function with Parameters (Arguments)

People (= programmers) tend to use the parameters and arguments as the same thing.

- **Parameter** is what is declared in the function.
- **Argument** is what is passed through when calling the function.

Don't worry if you use it wrong. My experience is that most do.

Let's have an example.

```
In [3]: def hello_name(name):
            print(f"Hello {name}, nice to meet you!")
```

Here **name** is the parameter.

```
In [4]: hello_name("Rune")
        Hello Rune, nice to meet you!
```

And **"Rune"** is the argument.

We see how you can pass arguments to the function and make it depend on the context you call.

```
In [5]: def print_age(name, age):
            print(f"Hello {name}")
            print(f"You are {age} years old")
```

As the example shows, you can have multiple lines in the function.

```
In [6]: print_age("Rune", 30)
        Hello Rune
        You are 30 years old
```

Returning Values from a Function

This is where the return statement comes into the picture.

```
In [7]: def calculate_age(birth_year):
            return 2021 - birth_year
```

Now, this could already be a useful function. Imagine that you make this calculation multiple times in your code. Then, something unexpected happens. A new year: 2022.

Then, you need to update it all over your code. And what if you forget in one place?

Using a function, you only need to update it in one place.

Copyright © 2023 learnpythonwithrune.org

```
In [8]: calculate_age(1991)
Out[8]: 30
```

A call to the function returns the value.

A Few Things for the Project

In the project, we will need to iterate over the characters in a string and change them.

We already know how to iterate over characters, but we need to learn about mapping.

Each character is represented by a numeric value called Unicode.

Let's try.

```
In [9]: ord('A')
Out[9]: 65
```

Hence, the value of **'A'** is 65. The value 65 is represented as an integer, which we would like.

Similarly, we can transform an integer value to the Unicode character it represents:

```
In [10]: chr(65)
Out[10]: 'A'
```

This is awesome.

Let's try to print the alphabet.

```
In [11]: for c in range(65, 65 + 26):
             print(f"{c}: {chr(c)}", end=', ')

65: A, 66: B, 67: C, 68: D, 69: E, 70: F, 71: G, 72: H, 73: I, 74: J, 75: K, 76: L, 77:
M, 78: N, 79: O, 80: P, 81: Q, 82: R, 83: S, 84: T, 85: U, 86: V, 87: W, 88: X, 89: Y, 9
0: Z,
```

Where I exploit **range(begin, end) and use end=', '** in the print statement to keep it on one line.

But you see that the capital letters have the values 65 to 90 (A to Z).

This makes it possible to map from one character to another.

We need to be able to check if a character is in a string.

```
In [12]: def check_char(c):
             if c in "ABC":
                 print(f"{c} is in ABC")
             else:
                 print(f"{c} is not in ABC")
```

Calling the function shows how it works.

```
In [13]: check_char('A')
A is in ABC
```

. . . and:

Copyright © 2023 learnpythonwithrune.org

```
In [14]: check_char('D')
         D is not in ABC
```

Similarly, you can check if a character is not in a string by using **not**.

```
In [15]: def check_char(c):
             if c not in "ABC":
                 print(f"{c} is not in ABC")
             else:
                 print(f"{c} is in ABC")
```

Remainder Calculations (%)

Finally, we need to calculate with remainders.

What?

I know if you are not familiar with that it sounds scary. I ensure you that it is not.

Let's look at another example.

Let's say you count the hours since midnight on new year.

After 1 hour, it is 1 a.m.

After 13 hours, it is 1 p.m.

After 25 hours, it is 1 a.m. again.

The calculations with modular arithmetic are the same as counting hours.

In this case, you would calculate modular 24 to get the time of day.

Say it is 25 hours since the new year.

25 % 24 = 1.

If 100 hours passed since the new year at midnight, then 100 % 24 = 4.

That means it is now 4 a.m.

Similarly, we can see if a number is even or odd.

0 % 2 = 0, even.

1 % 2 = 1, odd.

2 % 2 = 0, even.

3 % 2 = 1, odd.

And then it continues, just like the clock.

```
In [16]: def is_even(n):
             if n % 2 == 0:
                 print(f"{n} is even")
             else:
                 print(f"{n} is odd")
```

And two calls to the function.

Copyright © 2023 learnpythonwithrune.org

```
In [17]: is_even(16)
         16 is even
```

And.

```
In [18]: is_even(131)
         131 is odd
```


Copyright © 2023 learnpythonwithrune.org

08 – Dictionaries

In this chapter, we will learn about using **dictionaries** for programming.

Learning Objectives

- What is a **dictionary**?
- How to define and use **dictionaries**
- Add values to **dictionaries**
- How to iterate over **dictionaries**

What is a Dictionary?

A **Python dictionary** (**dict**) is similar to a word dictionary.

With a word dictionary, you identify a **word** you want to look up, and when you look up the **word,,** there is a **description**.

In a **Python dictionary** (**dict**), you have a **key** you want to look up. When you look up the **key**, you get the **value**.

Hence, **Python dictionaries** are composed of **key-values**. You can look up **values** by a **key**.

Your First Dictionary

It is often easier to understand when you see it.

```
In [1]: my_dict = {
            'Key 1': 'Value 1',
            'Key 2': 'Value 2'
        }
```

The **key** and **values** can be anything. Here, we use strings with simple content to make it easier to understand.

The **dictionary** looks like this.

```
In [2]: my_dict
Out[2]: {'Key 1': 'Value 1', 'Key 2': 'Value 2'}
```

You can look up a **value** with a **key**.

```
In [3]: my_dict['Key 2']
Out[3]: 'Value 2'
```

Insert New Key-Value pair in a Dictionary

Adding new **key-value** pair is straightforward.

Copyright © 2023 learnpythonwithrune.org

```
In [4]: my_dict['Key 3'] = 'Value 3'
```

This results in a dictionary looking like this.

```
In [5]: my_dict
Out[5]: {'Key 1': 'Value 1', 'Key 2': 'Value 2', 'Key 3': 'Value 3'}
```

Using Dictionaries as Records

There are many use-cases of **dictionaries**, but it is handy to keep a record of this. It's like an entry in your database.

```
In [6]: car = {
            "Brand": "Lamborghini",
            "Model": "Sián",
            "Year": 2020
        }
```

This results in the following.

```
In [7]: car
Out[7]: {'Brand': 'Lamborghini', 'Model': 'Sián', 'Year': 2020}
```

Using Dictionaries as Counters

Sometimes you need to count different values. This can be done with a **dictionary**.

```
In [8]: items = ['Pen', 'Scissor', 'Pen', 'Pen', 'Scissor']

        count = {}

        for item in items:
            count[item] = count.get(item, 0) + 1
```

In this case, we have a **list** of items, and we want to count the occurrences of each item type.

We create a **dictionary count**, which is empty.

We might not know what items are in the **list**, so we keep it empty before we **loop** over **it**.

Then for each item, we assign the current value added 1. This is done by using the **count.get(item, 0),** which returns the **value** of the **key** item or 0 if it does not exist.

If we have not seen a key before, say **'Pen,'** then it will return 0.

```
In [9]: count
Out[9]: {'Pen': 3, 'Scissor': 2}
```

As we can see, this counts all the items on the list **items**.

Copyright © 2023 learnpythonwithrune.org

Iterating over a Dictionary

As with **lists**, you can iterate over a **dictionary**. However, with **dictionaries,** you cannot do it directly. Instead, you can do this by calling a method on the dictionary, which will let you iterate over the **key-value** pairs, as the following example shows.

```
In [10]: for key, value in count.items():
             print(key, value)

Pen 3
Scissor 2
```


Copyright © 2023 learnpythonwithrune.org

09 – CSV Files

In this chapter, we will learn to read **CSV** files.

Learning Objectives

- What is a **CSV** file?
- How to read **CSV** files
- How to process data from a **CSV** file

What is a CSV file?

A comma-separated values (**CSV**) file is a delimited text file that uses a comma to separate values. Each line of the file is a data record.

Read that last sentence again. It sounds like each line can be a **dictionary**.

What does a CSV look like?

Let's first see what a **CSV** can look like.

```
First name, Last name, Age
Connar, Ward, 15
Rose, Peterson, 18
Paul, Cox, 12
Hanna, Hicks, 10

```

As you see here, the first line has the **key** (or column names), and each **row** has the corresponding **values** for the **keys**.

Please note that **CSV** files can be structured differently. For example, it can use a different delimiter **(,),** use quotes to capture the strings, etc..

How to read a CSV file

To do that, we need a library.

```
In [1]: import csv
```

Then, we can read the content as follows.

```
In [2]: with open("files/NameRecords.csv", "r") as f:
            csv_reader = csv.DictReader(f)
            name_records = list(csv_reader)
```

This is a bit new. We write **with open(…) as f:,** which will open the file and have a file pointer **f.**

Then we create a dictionary reader (**csv.DictReader(f)**).

Finally, as we will see here, we convert the **CSV reader** to a **list** of **dictionaries**.

Copyright © 2023 learnpythonwithrune.org

```
In [3]: name_records
Out[3]: [{'First name': 'Connar', 'Last name': 'Ward', 'Age': '15'},
         {'First name': 'Rose', 'Last name': 'Peterson', 'Age': '18'},
         {'First name': 'Paul', 'Last name': 'Cox', 'Age': '12'},
         {'First name': 'Hanna', 'Last name': 'Hicks', 'Age': '10'}]
```

As it is a **list** of **dictionaries**, you can access **rows** like you access a list and an entry.

```
In [4]: name_records[0]['First name']
Out[4]: 'Connar'
```

Process Content from a CSV file

This can be done by iterating over the **list** of **dictionaries**, as the example below shows.

```
In [5]: for name_record in name_records:
            print(f"{name_record['First name']} is {name_record['Age']} years")

Connar is 15 years
Rose is 18 years
Paul is 12 years
Hanna is 10 years
```


Copyright © 2023 learnpythonwithrune.org

10 – Recursion

In this chapter, we will learn about **recursion**.

Learning Objectives

- What is a **recursion**?
- Why use **recursion**?
- How to use **recursion** to solve a problem

What is Recursion?

If a function calls itself to solve the problem, it is called **recursion**.

What does that mean?

Well, you can probably relate to the following. Often when you solve a big problem, you can divide it down into similar smaller problems and solve them first, then based on these results, you get the total result.

Imagine that you need to count money. You do that by dividing the money into smaller groups, counting each group's money, and then adding the results together.

That is **recursion**.

Why use Recursion?

Recursion by itself does not enable you to solve more problems, but it helps you to solve problems more easily.

Recall counting money by dividing it into smaller groups.

This does not enable you to count money, but it might make it easier if you lose track of a count. If you counted all the money at once, you would need to start all over. If you had grouped the money, you might only need to re-count one group.

You can think of **recursion** with programming in a similar manner. It makes it easier to understand the problem and makes the code more digestible for the reader.

The first time you see **recursive** code, you might not think it makes it easier.

The First Rule of Recursion

You need a base case to terminate.

Consider the following code.

```
In [1]: def count(n):
            return n + count(n - 1)
```


Copyright © 2023 learnpythonwithrune.org

What happens if you call the code?

Well, it will terminate with an error: **RecursionError:** *maximum recursion depth exceeded.*

As you see, it keeps calling itself and will never end calling itself.

Hence, the first rule you need to understand with **recursion** is you need a base-case, which will not call itself anymore.

```
In [1]: def count(n):
            if n == 0:
                return 0
            return n + count(n - 1)
```

And if we call it:

```
In [2]: count(6)
Out[2]: 21
```

You can think of it like this.

count(2) = 2 + count(1) = 2 + 1 + count(0) = 2 + 1 + 0 = 3

Feel free to do it for 6, and you will get 21.

Fibonacci Sequence

The Fibonacci sequence is defined recursively.

Given the Fibonacci number defined as

- $F_0 = 0$
- $F_1 = 1$
- $F_n = F_{n-1} + F_{n-2}$

This should give a sequence: 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, and so forth.

Let's try. Start by defining the base case.

```
In [3]: f0 = 0
        f1 = 1
```

Then we calculate.

```
In [4]: f2 = f1 + f0
        f2
Out[4]: 1
```

That looks good.

```
In [5]: f3 = f2 + f1
        f3
Out[5]: 2
```

That continues to look good.

Copyright © 2023 learnpythonwithrune.org

```
In [6]: f4 = f3 + f2
        f4
Out[6]: 3
```

Honest, I am confident that you can continue on your own.

But there is a problem with this approach. It doesn't feel like the most efficient way to calculate it.

So, try this instead, which applies the principles of recursion.

```
In [7]: def fib(n):
            if n < 2:
                return n
            return fib(n - 1) + fib(n - 2)
```

And we can continue.

```
In [8]: fib(5)
Out[8]: 5
```

And more.

```
In [9]: fib(6)
Out[9]: 8
```

And the rest is for you to enjoy.

Copyright © 2023 learnpythonwithrune.org

11 – Comprehension

In this chapter, we will learn about **list** and **dictionary comprehension**.

Learning Objectives

- What is **comprehension**?
- How to use **list comprehension**
- How to use **dictionary comprehension**

What is Comprehension?

List comprehension offers a shorter syntax when you want to create a new **list** based on the values of an existing list, which is similar to **dictionaries**.

That is, you can create them in one line instead of using a **loop** to create them.

The best way to learn about them is to see how it works.

Create your First List Comprehension

You can create a list from a range.

```
In [1]: my_list = [i for i in range(10)]
        my_list
Out[1]: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

This code is similar to the following.

```
In [2]: my_list = []

        for i in range(10):
            my_list.append(i)

        my_list
Out[2]: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

In the **list comprehension** (cell 1), we wrote the **for**-loop inside the list, while a traditional way would have an outside **for**-loop.

Create a List from a List using Comprehension

As noted in the description of **list comprehension**, you can create a **list** from another **list**.

```
In [3]: another_list = [i*i for i in my_list]
        another_list
Out[3]: [0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
```

This could also be achieved with a traditional loop.

Copyright © 2023 learnpythonwithrune.org

```
In [4]: another_list = []

        for i in my_list:
            another_list.append(i*i)

        another_list
```

```
Out[4]: [0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
```

List Comprehension with If-statement

Say you only need some elements based on an **if**-statement.

Luckily**, list comprehension** can fix that for you, too.

```
In [5]: even_list = [i for i in my_list if i % 2 == 0]
        even_list
```

```
Out[5]: [0, 2, 4, 6, 8]
```

The **if**-statement is added at the end.

The syntax changes a bit if you add an **else**-statement.

```
In [6]: even_list = [i if i % 2 == 0 else -i*i for i in my_list]
        even_list
```

```
Out[6]: [0, -1, 2, -9, 4, -25, 6, -49, 8, -81]
```

But it works.

Dictionary Comprehension

As you will see, this is similar.

```
In [7]: {i: i*i for i in my_list}
```

```
Out[7]: {0: 0, 1: 1, 2: 4, 3: 9, 4: 16, 5: 25, 6: 36, 7: 49, 8: 64, 9: 81}
```

From a **list,** you can create a **dictionary**.

Copyright © 2023 learnpythonwithrune.org

12 – Object-Oriented Programming

In this chapter, we will learn about **object-oriented programming** (**OOP**).

Learning Objectives

- What is **object-oriented programming**?
- How to define **classes**
- Normal **methods** to declare

What is Object-Oriented Programming?

At its core, **object-oriented programming** helps you structure your program to resemble reality. You declare objects with parameters and methods you need on them.

In the project, we will make a card game. The objects we will create are **Card**, **Deck**, and **Hand**. The **Card** will represent a card with a suit and rank. The **Deck** will be a list of **Card**s, and you will be able to add a card, remove a card, and shuffle the cards. The **Hand** will be like a **Deck**and belonging to a player to keep track of wins.

This makes the programming easier as it relates to something we know.

When using **object-oriented programming,** try to think how you would do it without declaring classes.

Class and Objects: What is the Difference?

A **class** is the blueprint of an **object**.

Hence, the **object** is the **instances** of the **class**.

The **class** describes the **attributes** of the **class** and the **methods** you can **call** on the **object** once created.

But let's become a bit more concrete and explain along the way.

How to Declare an Object

Let's declare our first **class**, blueprint of the card **object**.

```
In [1]: class Card:
            def __init__(self, suit, rank):
                self.suit = suit
                self.rank = rank
```

The **class Card** defines the name of the class.

The **def __init__(self, suit, rank):** defines the initialization method of the class.

Copyright © 2023 learnpythonwithrune.org

The **self** is crucial to have for all the methods we declare. If you assign anything to **self**, you can access it from anywhere in the class.

Let's continue.

```
In [2]: card = Card(0, 0)
```

We create a **card** instance here. That means we have an object **Card** assigned to the **card**.

Let's try to see it.

```
In [3]: card
Out[3]: <__main__.Card at 0x7fa9f4487460>
```

This was not what we expected?

As we will see, we can define how an object should be represented as a string.

Object String Method

If we define a **__str__(self)** method, we can print it.

```
In [4]: class Card:
            def __init__(self, suit, rank):
                self.suit = suit
                self.rank = rank

            def __str__(self):
                return f'{self.suit} {self.rank}'
```

Notice how we indent the code.

Let's try again.

```
In [5]: card = Card(1, 2)

In [6]: card
Out[6]: <__main__.Card at 0x7fcfc7b909d0>
```

What?

Oh, we need to call **print** or **str** to get the representation.

```
In [7]: print(card)
        1 2
```

Or

```
In [8]: str(card)
Out[8]: '1 2'
```

Let's make it a bit more appealing.

Copyright © 2023 learnpythonwithrune.org

```
In [9]: class Card:
            suits = ['\u2666', '\u2665', '\u2663', '\u2660']
            ranks = ["2", "3", "4", "5", "6", "7", "8", "9", "10", "J", "Q", "K", "A"]

            def __init__(self, suit, rank):
                self.suit = suit
                self.rank = rank

            def __str__(self):
                return f'{Card.suits[self.suit]}{Card.ranks[self.rank]}'
```

We use Unicode values for the suits and declare the card values with **J, Q, K,** and **A**. This makes the value ranked.

```
In [10]: card = Card(2, 12)
```

Let's see what it looks like.

```
In [11]: str(card)
Out[11]: '♣A'
```

Now that's nice.

Making Objects Comparable

This can be done by adding a method **__lt__(self, other):**

```
In [12]: class Card:
             suits = ['\u2666', '\u2665', '\u2663', '\u2660']
             ranks = ["2", "3", "4", "5", "6", "7", "8", "9", "10", "J", "Q", "K", "A"]

             def __init__(self, suit, rank):
                 self.suit = suit
                 self.rank = rank

             def __str__(self):
                 return f'{Card.suits[self.suit]}{Card.ranks[self.rank]}'

             def __lt__(self, other):
                 t1 = self.rank, self.suit
                 t2 = other.rank, other.suit
                 return t1 < t2
```

Let's see how this works.

```
In [13]: card1 = Card(2, 12)
         card2 = Card(1, 10)
```

We are creating two objects.

```
In [14]: print(card1, card2)
         ♣A ♥Q
```

Then we will compare them.

```
In [15]: card1 > card2
Out[15]: True
```

This demonstrates why we have **A** (ace) at the end of our list of ranks.

Copyright © 2023 learnpythonwithrune.org

```
In [16]: card1 < card2
Out[16]: False
```

Creating the Deck Class

We want the **Deck** to create a pile of all the cards.

```
In [17]: class Deck:
             def __init__(self):
                 self.cards = []
                 for suit in range(4):
                     for rank in range(13):
                         card = Card(suit, rank)
                         self.cards.append(card)
```

If you want to get the length of a class, you can add method **__len__(self)**

```
In [18]: class Deck:
             def __init__(self):
                 self.cards = []
                 for suit in range(4):
                     for rank in range(13):
                         card = Card(suit, rank)
                         self.cards.append(card)

             def __len__(self):
                 return len(self.cards)
```

We just use the length of the list. Makes sense, right?

```
In [19]: deck = Deck()
         len(deck)
Out[19]: 52
```

Obviously, you can return whatever you want. But the length should be related to what the object represents. Here is a deck of cards, initially with 52 cards.

We would like to be able to take a card and add a card.

```
In [22]: class Deck:
             def __init__(self):
                 self.cards = []
                 for suit in range(4):
                     for rank in range(13):
                         card = Card(suit, rank)
                         self.cards.append(card)

             def __len__(self):
                 return len(self.cards)

             def add_card(self, card):
                 self.cards.append(card)

             def pop_card(self):
                 return self.cards.pop()
```

As an example:

Learn Python with Rune

Copyright © 2023 learnpythonwithrune.org

```
In [23]: deck = Deck()
         print(f"Number of cards: {len(deck)}")
         card = deck.pop_card()
         print(f"Number of cards: {len(deck)}")
         deck.add_card(card)
         print(f"Number of cards: {len(deck)}")
```

```
Number of cards: 52
Number of cards: 51
Number of cards: 52
```

Finally, we would like to shuffle the card.

The random library can do that for us. Remember?

```
In [24]: import random
```

We add a shuffle method and call it from **__init__(self)**.

```
In [25]: class Deck:
             def __init__(self):
                 self.cards = []
                 for suit in range(4):
                     for rank in range(13):
                         card = Card(suit, rank)
                         self.cards.append(card)
                 self.shuffle()

             def __len__(self):
                 return len(self.cards)

             def add_card(self, card):
                 self.cards.append(card)

             def pop_card(self):
                 return self.cards.pop(i)

             def shuffle(self):
                 random.shuffle(self.cards)
```

Create the Hand Class

A great concept in **object-oriented programming** is inheritance.

That means you can create a class that gets the same methods and attributes as another class.

Let's demonstrate.

```
In [26]: class Hand(Deck):
             def __init__(self, label):
                 self.cards = []
                 self.label = label
                 self.win_count = 0
```

The **class Hand(Desk)** declares that Hand will have the same methods as **Deck**.

In this case, we overwrite the original **__init__**. Hence, this will run on creation. So far, this is the only difference.

```
In [27]: hand = Hand('Player 1')
```

As you see, now we pass a label as an argument.

Copyright © 2023 learnpythonwithrune.org

```
In [28]: hand.add_card(card)
```

But we can apply methods declared in **Deck**.

If we want to see the content of the hand, we can declare the **__str__(self)** method.

```
In [29]: class Hand(Deck):
             def __init__(self, label):
                 self.cards = []
                 self.label = label
                 self.win_count = 0

             def __str__(self):
                 return self.label + ": " + ' '.join([str(card) for card in self.cards])
```

Here is an example:

```
In [33]: hand = Hand('Player 1')
```

```
In [34]: hand.add_card(deck.pop_card())
         hand.add_card(deck.pop_card())
```

```
In [35]: print(hand)
```

```
Player 1: ♠A ♠K
```

In addition, we would like to be able to update the score.

```
In [36]: class Hand(Deck):
             def __init__(self, label):
                 self.cards = []
                 self.label = label
                 self.win_count = 0

             def get_label(self):
                 return self.label

             def round_winner(self):
                 self.win_count += 1

             def get_win_count(self):
                 return self.win_count

             def __str__(self):
                 return self.label + ": " + ' '.join([str(card) for card in self.cards])
```

Now we are ready to create our first card game.

Learn Python with Rune

Copyright © 2023 learnpythonwithrune.org

13 – Matplotlib

In this chapter, we will learn how to visualize charts with **b**.

Learning Objectives

- What is **Matplotlib**?
- How to make **charts**
- The **object-oriented** way and the **functional** form of using **Matplotlib**

What is Matplotlib?

Matplotlib is a comprehensive library for creating static, animated, and interactive visualizations in **Python**.

Your First Visualization

When you use **Matplotlib** in **Jupyter** Notebook, you need to import it, as you do here.

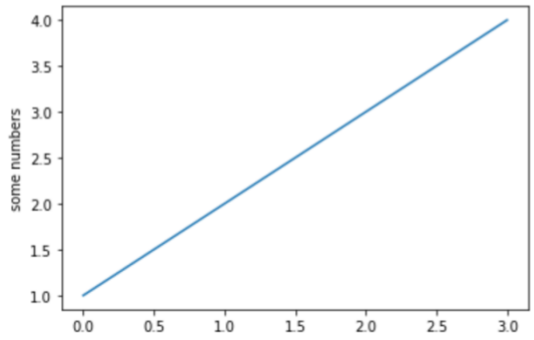
```
In [1]: import matplotlib.pyplot as plt
        %matplotlib inline
```

The **%matplotlib inline** tells **Jupyter** Notebook how to render the image. The **inline** is a static rendering. If you want an interactive, try **notebook** instead.

Here we will use the static rendering.

Let's try our first plot.

```
In [2]: plt.plot([1, 2, 3, 4])
        plt.ylabel('some numbers')
Out[2]: Text(0, 0.5, 'some numbers')
```



This is the functional way to use **Matplotlib**. However, I have found it can be a bit confusing to use it like this. Therefore, I will show the **object-oriented** way, which requires a line more, but it will keep you from making mistakes later.

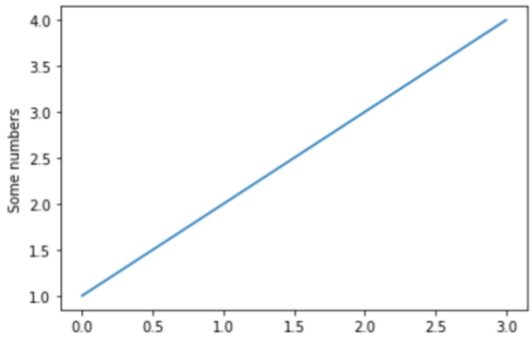
Copyright © 2023 learnpythonwithrune.org

Creating Plot using Object-Oriented Way

This requires one line extra.

```
In [3]: fig, ax = plt.subplots()
        ax.plot([1, 2, 3, 4])
        ax.set_ylabel("Some numbers")
```

```
Out[3]: Text(0, 0.5, 'Some numbers')
```



Here we create a figure (**fig**) and an axis (**ax**).

The **object-oriented** way makes it easy to understand which figure and axis we draw on. The **functional** way often makes it confusing, and you might make errors.

Scatter Chart

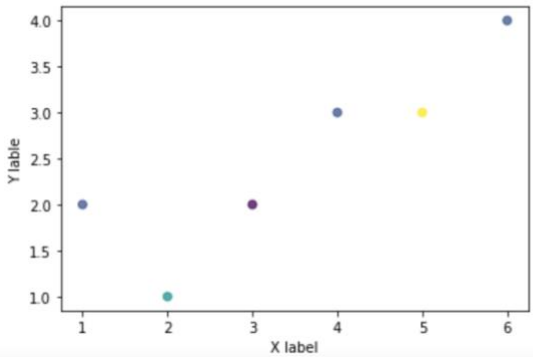
Let's make some random data.

```
In [4]: x = [1, 2, 3, 4, 5, 6]
        y = [2, 1, 2, 3, 3, 4]
        c = [2, 3, 1, 2, 5, 2]
```

And try to scatter it.

```
In [5]: fig, ax = plt.subplots()
        ax.scatter(x, y, c=c, alpha=.75)
        ax.set_xlabel("X label")
        ax.set_ylabel("Y lable")
```

```
Out[5]: Text(0, 0.5, 'Y lable')
```



Notice that we have used colors. They are set by the list **c** and argument **c=c.** The argument **alpha=.75** makes it a bit transparent.

Copyright © 2023 learnpythonwithrune.org

Histogram Chart

Again, let's make some random data.

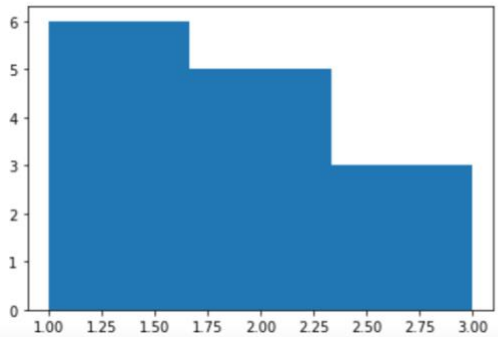
```
In [6]: data = [1, 2, 2, 3, 2, 1, 1, 1, 2, 3, 2, 3, 1, 1]
```

And a histogram.

```
In [7]: fig, ax = plt.subplots()

        ax.hist(data, bins=3)

Out[7]: (array([6., 5., 3.]),
         array([1.        , 1.66666667, 2.33333333, 3.        ]),
         <BarContainer object of 3 artists>)
```



Notice we set the **bins=3** to make three bins for the data in the histogram.

Learn Python with Rune

Copyright © 2023 learnpythonwithrune.org

14 – NumPy and Linear Regression

In this chapter, we will learn to use **Linear Regression**.

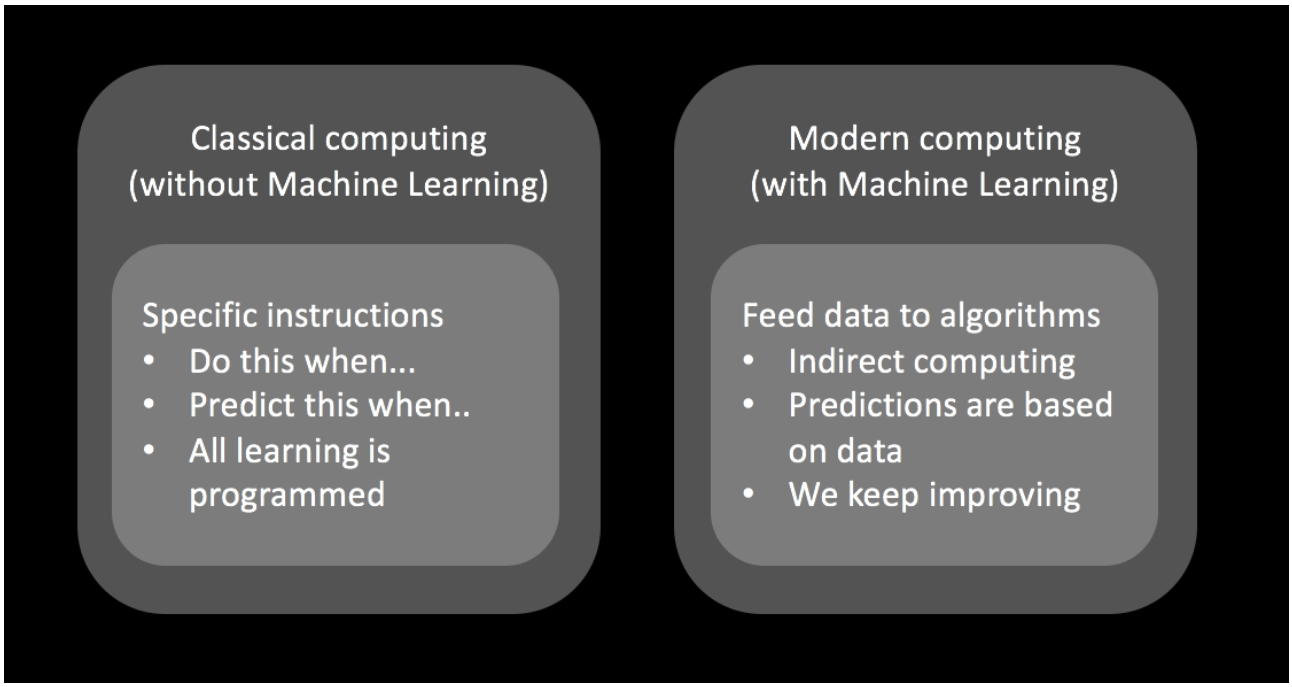
Learning Objectives

- What is **Machin Learning**?
- What is **Linear Regression**?
- What is **NumPy** and how to use it
- How to make a **model** and **fit** it
- How to **visualize** the **result**

What is Machine Learning?

In the **classical computing** model, everything is programmed into **algorithms**. This has the limitation that all **decision logic** needs to be **understood before usage**. And if **things change**, we need to **modify the program**.

With the **modern computing** model (**Machine Learning**), this paradigm changes. We **feed** the **algorithms** with **data**. Based on that data, we **make** the **decisions** in the **program**.



How Machine Learning Works

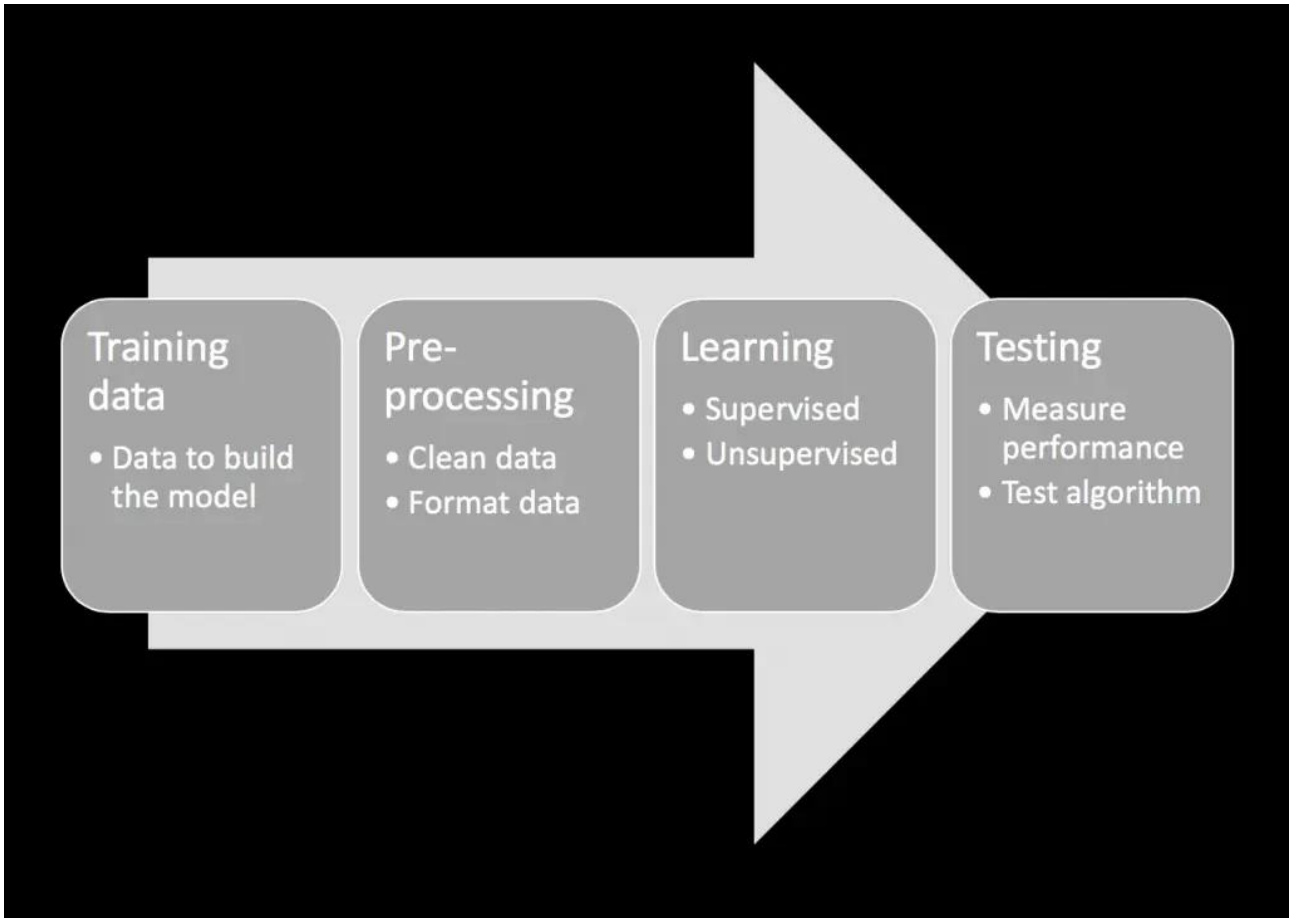
On a high level, we can divide **Machine Learning** into two phases:

- **Phase 1**: Learning
- **Phase 2**: Prediction

Learn Python with Rune

Copyright © 2023 learnpythonwithrune.org

The **learning phase** (Phase 1) can be divided into sub-steps.



It all starts with a **training set** (training data). This data set should represent the type of data that the **Machine Learning** model should be used to predict in Phase 2 (prediction).

The **pre-processing** step is about **cleaning up data**. While **Machine Learning** is awesome, it cannot figure out what good data looks like. You need to clean and transform data into the desired format.

Then for the magic step: learning. There are three main machine learning paradigms:

- **Supervised Learning**: you tell the algorithm what categories each data item is in. Each data item from the training set is tagged with the right answer.
- **Unsupervised Learning**: the learning algorithm is not told what to do with it and should make the structure itself.
- **Reinforcement Learning**: teaches the machine to think for itself based on past action rewards.

Finally, if the model is good, the testing is done. The training data was divided into a test set and a training set. We use the test set to see if the model can predict from it. If not, a new model might be necessary.

Then the prediction begins.

Copyright © 2023 learnpythonwithrune.org

What is Linear Regression?

Linear regression is a approach to modelling the relationship between a scalar response to one or more variables. If we try to model, we will do it for one single variable. In another way, we want map points on a graph to a line (**y = a*x + b**).

What is NumPy and a Quick Introduction

NumPy is the fundamental package for **scientific computing** with **Python**.

A **NumPy** array is a data structure used for scientific computing. It is similar to a **Python list**, just with some restrictions.

- A **NumPy** array only has **one type** for all entries.
- If **multidimensional**, all rows must be of the **same shape**.
- You **cannot change** the **size** dynamically.

Why restrictions?

It makes it 10 to 100 times faster than **Python lists**.

To use them, we need to import a library.

```
In [1]: import numpy as np
```

We can create **NumPy** arrays from **Python lists**.

```
In [2]: a1 = np.array([1, 2, 3, 4])
        a2 = np.array([5, 6, 7, 8])
```

We call them arrays, as they are similar to **C** arrays.

```
In [3]: a1
Out[3]: array([1, 2, 3, 4])
```

They have a **shape**.

```
In [4]: a1.shape
Out[4]: (4,)
```

They only have one data type for all entries.

```
In [5]: a1.dtype
Out[5]: dtype('int64')
```

You can make simple arithmetic directly with them.

```
In [6]: a1 + a2
Out[6]: array([ 6,  8, 10, 12])
```

Multiply by a scalar.

Copyright © 2023 learnpythonwithrune.org

```
In [7]: a1*2
Out[7]: array([2, 4, 6, 8])
```

Multiply them entry-wise.

```
In [8]: a1*a2
Out[8]: array([ 5, 12, 21, 32])
```

This is enough to get us started.

Creating a Simple Linear Regression Model

We need to import the library containing the model.

```
In [9]: from sklearn.linear_model import LinearRegression
        import matplotlib.pyplot as plt
        %matplotlib inline
```

We also import **Matplotlib**, as we will use it to visualize it.

To demonstrate how it works, we will use simple data.

```
In [10]: x = [i for i in range(10)]
         y = [i for i in range(10)]
```

```
In [11]: print(x, y)
         [0, 1, 2, 3, 4, 5, 6, 7, 8, 9] [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

Why do we need **NumPy**? Because the model takes the data in a specific format.

```
In [12]: X = np.array(x).reshape((-1, 1))
         Y = np.array(y).reshape((-1, 1))
```

The **reshape(-1, 1)** turns the data from a single row to a single column.

```
In [13]: X
Out[13]: array([[0],
                [1],
                [2],
                [3],
                [4],
                [5],
                [6],
                [7],
                [8],
                [9]])
```

Then we can create the model.

```
In [14]: lin_regressor = LinearRegression()
         lin_regressor.fit(X, Y)
         Y_pred = lin_regressor.predict(X)
```

Where we also get the predictions.

Notice that we use all the data and do not divide it into training and test data.

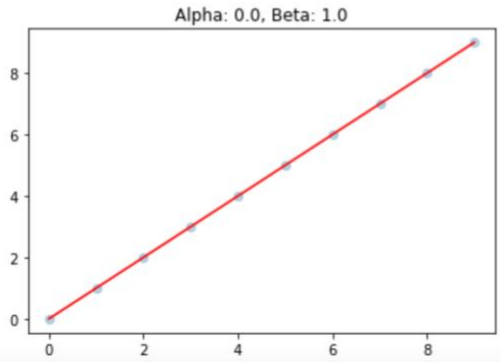
Copyright © 2023 learnpythonwithrune.org

To visualize the data, we need the coefficients of the linear regression model.

```
In [15]: alpha = str(round(lin_regressor.intercept_[0], 5))
         beta = str(round(lin_regressor.coef_[0][0], 5))
```

Just to put them on the fitted chart.

```
In [16]: fig, ax = plt.subplots()
         ax.set_title("Alpha: " + alpha + ", Beta: " + beta)
         ax.scatter(X, Y, alpha=0.3)
         ax.plot(X, Y_pred, c='r')
Out[16]: [<matplotlib.lines.Line2D at 0x7ff420d5a520>]
```



That is the process you need for the project.

Learn Python with Rune

Copyright © 2023 learnpythonwithrune.org

15 – Pandas and Export to Excel

In this chapter, we will learn to use **Pandas** to enrich **CSV** file data and export it to an **Excel** sheet.

Learning Objectives

- What is **Pandas**?
- How to work with **DataFrames** (Pandas data structure)
- Export data to **Excel** sheet from **Pandas**
- How to enrich with **charts** in **Excel** from **Python**

What is Pandas?

Pandas is a fast, powerful, flexible, and easy-to-use, open-source data analysis and manipulation tool built on top of the **Python** programming language.

Pandas' main data structure, **DataFrame**, is quite similar to an **Excel sheet**. Just with a few restrictions.

What is a DataFrame?

A **DataFrame** is a two-dimensional data structure with columns of data. Each column can have its own data type. Also, **DataFrames** have an index as well as labels for the columns.

A DataFrames is like a spreadsheet in many aspects. It is generally the most commonly used **Pandas** object.

Get Started with Pandas

We will use **Pandas** to read **CSV** files and work with the data in the **DataFrame**.

To work with **Pandas DataFrames** we need to import **pandas**.

```
In [1]: import pandas as pd
```

Then we can read data directly from a **CSV** file into a **DataFrame**.

```
In [2]: sales_data = pd.read_csv("files/SalesData.csv",
                                 index_col='Date',
                                 delimiter=';',
                                 parse_dates=True)
```

A few arguments.

- **index_col**: Sets which column to be used as an index. Here we use **Date**.
- **delimiter**: If the default comma (**,**) is not used, you must specify it. This **CSV** file uses **;**
- **parse_dates**: If not set to **True**, any dates will be interpreted as strings (or objects in DataFrames).

Copyright © 2023 learnpythonwithrune.org

To inspect the data, simply call **head()** on it, and it will show the top five rows.

```
In [3]: sales_data.head()
```

Out[3]:

Date	Sales rep	Item	Price	Quantity	Sale
2020-05-31	Mia	Markers	4	1	4
2020-02-01	Mia	Desk chair	199	2	398
2020-09-21	Oliver	Table	1099	2	2198
2020-07-15	Charlotte	Desk pad	9	2	18
2020-05-27	Emma	Book	12	1	12

As you see, this looks similar to a spreadsheet.

To see the data types of each column.

```
In [4]: sales_data.dtypes
```

```
Out[4]: Sales rep    object
        Item         object
        Price         int64
        Quantity      int64
        Sale          int64
        dtype: object
```

As you can see, the columns **Sales rep** and **Item** are objects (strings), and the rest of the integers (int64).

A Few Tricks with Pandas

If you want to know how much each sales representative sold, you can do that by grouping the data.

```
In [5]: repr = sales_data.groupby("Sales rep")
        repr_sales = repr['Sale'].sum()
```

Then you can see the results here.

```
In [6]: repr_sales
```

```
Out[6]: Sales rep
        Charlotte     74599
        Emma          65867
        Ethan         40970
        Liam          66989
        Mia           88199
        Noah          78575
        Oliver        89355
        Sophia       103480
        William       80400
        Name: Sale, dtype: int64
```

If you want to see the monthly sale.

```
In [7]: months = sales_data.groupby(pd.Grouper(freq="M"))
        months_sales = months['Sale'].sum()
        months_sales.index = months_sales.index.month_name()
```

And the result.

Copyright © 2023 learnpythonwithrune.org

```
In [8]: months_sales
```

```
Out[8]: Date
        January      69990
        February     51847
        March        67500
        April        58401
        May          40319
        June         59397
        July         64251
        August       51571
        September    55666
        October      50093
        November     57458
        December     61941
        Name: Sale, dtype: int64
```

Export to Excel

A simple export to **Excel** can be done as follows:

```
In [9]: writer = pd.ExcelWriter("SalesReport.xlsx")
        repr_sales.to_excel(writer, sheet_name="Sale per rep")
        months_sales.to_excel(writer, sheet_name="Sale per month")
        writer.close()
```

You create a **writer**, and write the two sheets.

It creates an **Excel** sheet in your working directory called **SalesReport.xlsx**.

If you want to enrich it with charts, you can do so accordingly:

```
In [10]: writer = pd.ExcelWriter("SalesReport.xlsx")
         repr_sales.to_excel(writer, sheet_name="Sale per rep")
         months_sales.to_excel(writer, sheet_name="Sale per month")

         chart = writer.book.add_chart({'type': 'column'})
         chart.add_series({
             'values': '=\'Sale per rep\'!$B$2:$B$10',
             'categories': '=\'Sale per rep\'!$A$2:$A$10',
             'name': "Sale"
         })
         writer.sheets['Sale per rep'].insert_chart("D2", chart)

         chart = writer.book.add_chart({'type': 'column'})
         chart.add_series({
             'values': '=\'Sale per month\'!$B$2:$B$13',
             'categories': '=\'Sale per month\'!$A$2:$A$13',
             'name': "Sale"
         })
         writer.sheets['Sale per month'].insert_chart("D2", chart)

         writer.close()
```


Learn Python with Rune

Copyright © 2023 learnpythonwithrune.org

Pandas Cheat Sheet

A great resource to remember all the **Pandas** methods is by using the **Pandas Cheat Sheet**.

You can download directly the Pandas Cheat Sheet here.

16 – Capstone Project

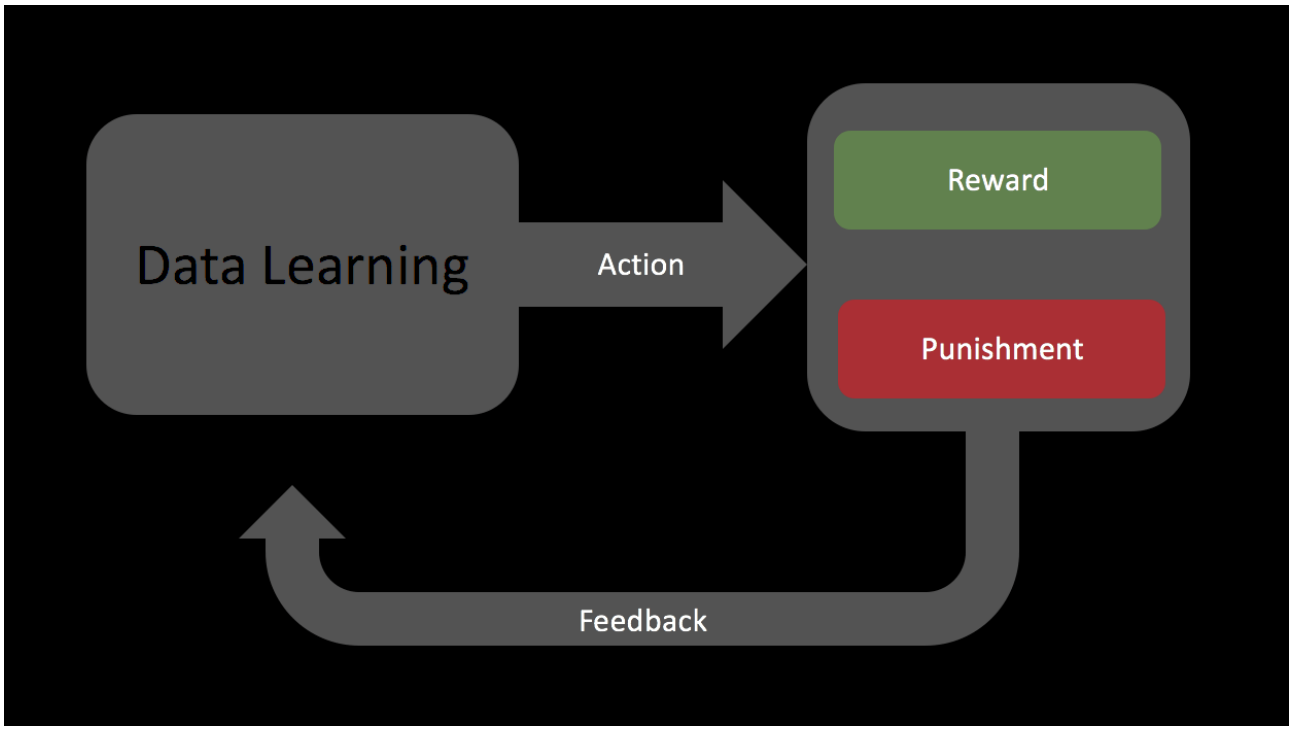
In this chapter, we will create our **Capstone Project** with **Reinforcement Learning**.

Learning Objectives

- What is **Reinforcement Learning**?
- A **brief introduction** to the problem
- The simple solution to the problem

What is Reinforcement Learning?

Reinforcement Learning teaches the machine to think for itself based on past action rewards.



Basically, the **Reinforcement Learning** algorithm tries to predict actions that give rewards and avoid punishment.

It is like training a dog. You and the dog do not talk the same language, but the dog learns how to act based on rewards (and punishment, which I do not advise).

Hence, if a dog is rewarded for a certain action in a given situation, then the next time it is exposed to a similar situation, it will act the same.

Copyright © 2023 learnpythonwithrune.org

Translate that to **Reinforcement Learning**.

- The **agent** is the dog that is exposed to the **environment**.
- Then the **agent** encounters a **state**.
- The **agent acts** to transition to a **new state**.
- Then after the transition, the **agent** receives a **reward** or **penalty** (punishment).
- This forms a **policy** to create a strategy to choose actions in a given **state**.

What Algorithms are Used for Reinforcement Learning?

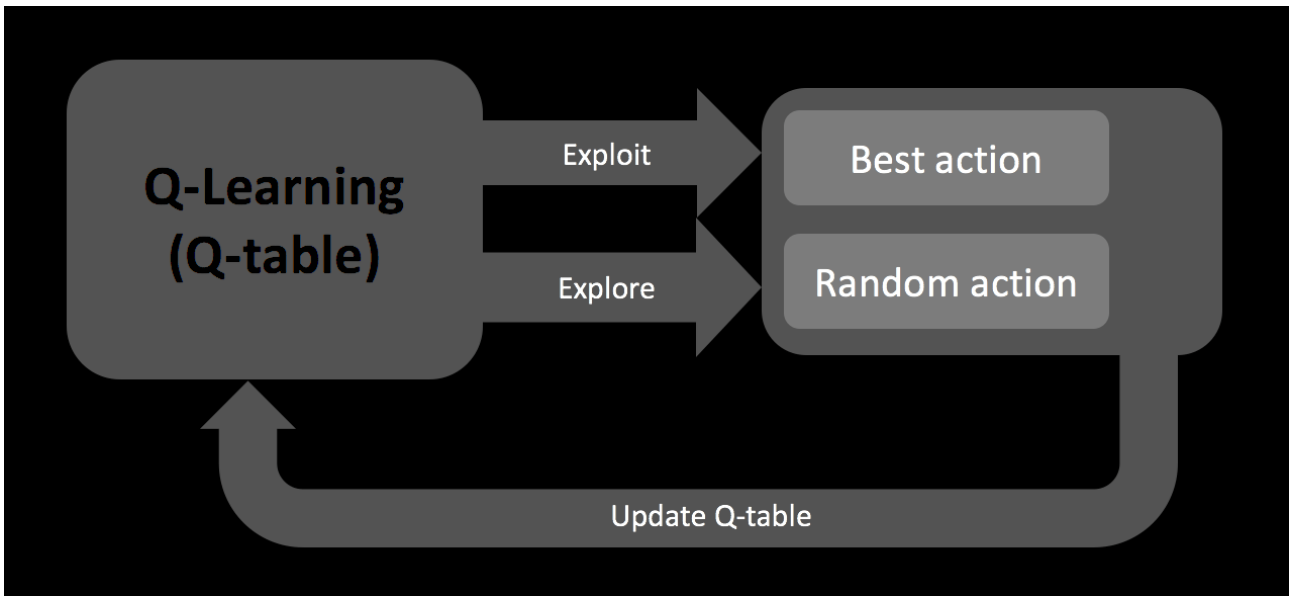
The most common algorithms for **reinforcement learning** are:

- **Q-Learning**, a model-free reinforcement learning algorithm to learn a policy telling an agent what action to take under what circumstances.
- **Temporal Difference** refers to a class of model-free reinforcement learning methods that learn by bootstrapping from the current estimate of the value function.
- **Deep Adversarial Network** is a technique employed in the field of machine learning which attempts to fool models through malicious input.

We will focus on the **Q-learning** algorithm as it is easy to understand and a powerful tool.

How does Q-learning Work?

As already noted, I just love this algorithm. It is *"easy"* to understand and powerful, as you will see.



- The **Q-Learning** algorithm has a **Q-table**, a matrix of dimension state x actions. Note: don't worry if you do not understand what a matrix is. You will not need the mathematical aspects of it. It is just an indexed "container" with numbers.
- The **agent** (or **Q-Learning** algorithm) will be in a **state**.
- Then in each iteration, the **agent** needs to take **action**.
- The **agent** will continuously update the **reward** in the **Q-table**.

Copyright © 2023 learnpythonwithrune.org

- The learning can come from either **exploiting** or **exploring**.
- This translates into the following **pseudo** algorithm for **Q-Learning**.
- The **agent** is in a given **state** and needs to choose an **action**.

The Algorithm

- Initialize the **Q-table** to all zeros
- Iterate
    - The **agent** is in **state**.
    - With probability, **epsilon** chooses to **explore**, else **exploit**.
        - If you choose to **explore**, then choose a *random* **action**.
        - If you choose to **exploit**, then choose the *best* **action** based on the current **Q-table**.
    - Update the **Q-table** from the new **reward** to the previous state.
    - Q[**state**, **action**] = (1 – **alpha**) * Q[**state**, **action**] + **alpha** * (**reward** + **gamma** * max(Q[**new_state**]) — Q[**state**, **action**])

Variables

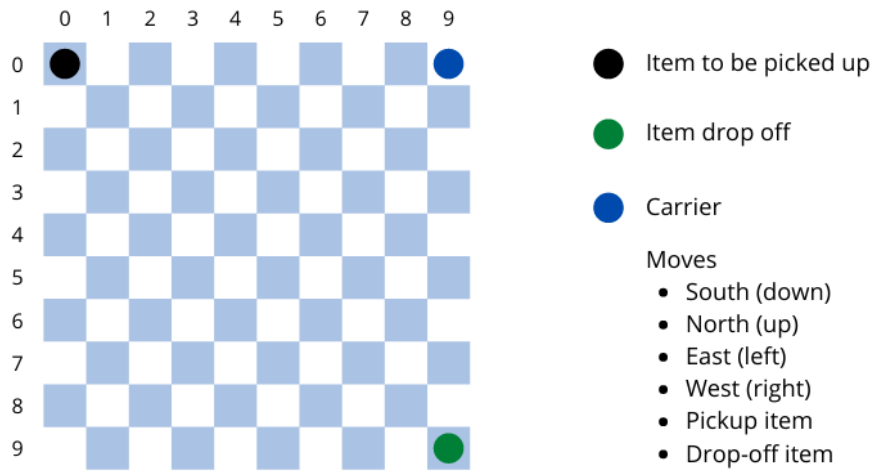
As you can see, we have introduced the following variables:

- **Epsilon**: the probability of taking a random action, which is done to explore new territory.
- **Alpha**: the learning rate that the algorithm should make in each iteration and should be in the interval from 0 to 1.
- **Gamma**: the discount factor used to balance the immediate and future reward. This value is usually between 0.8 and 0.99
- **Reward**: the feedback on the action and can be any number. Negative is penalty (or punishment), and positive is a reward.

Description of Capstone Project

To keep it simple, we create a **field** of size 10×10 positions. In that **field,** an item needs to be picked up and moved to a drop-off point.

Copyright © 2023 learnpythonwithrune.org



At each position, there are six different **actions** that can be taken.

- **Action 0**: Go south if on the field.
- **Action 1**: Go north if on the field.
- **Action 2**: Go east if on the field.
- **Action 3**: Go west if on the field.
- **Action 4**: Pickup item (it can try even if it is not there)
- **Action 5**: Drop-off item (it can try even if it does not have it)

Based on these actions, we will make a reward system.

- If the **agent** tries to go off the **field**, punish with **-10 in reward**.
- If the **agent** makes a (**legal**) **move**, punish with **-1 in reward**, as we do not want to encourage endless walking around.
- If the **agent** tries to **pick up** the item, but it is not there or has it already, punish with **-10**.
- If the **agent** picks up the item **correct place**, **reward with 20**.
- If the **agent** tries to **drop off** the item in the **wrong place** or does not have the item, **punish with -10**.
- If the **agent drops off** the item in the **correct place**, **reward it with 20**.

Solution?

See the full code on **GitHub**.

Copyright © 2023 learnpythonwithrune.org

Next Steps – Free Video Courses

Expert Data Science Blueprint

Data Science with Python is a 12+ hours FREE course – a journey from zero to mastery.

This is a 12-hours full **Expert Data Science** course. We focus on getting you started with **Data Science** with the most **efficient tools** and the proper understanding of **what adds value** to a **Data Science Project**.

Most use too much time to cover **too many technologies** without adding value and create poor quality Data Science Project. **You don't want to end up like that!** Follow the **Secret Data Science Blueprint**, which will give a **focused template** covering all you need to create **successful Data Science Projects**.

Read more here, get the projects, and see the videos.

Master Machine Learning Without Any High-Level Degree

Copyright © 2023 learnpythonwithrune.org

Most people believe that Machine Learning requires:

- A strong statistical background
- University level mathematics
- Proficiency in computer science

Agree?

No! That is a misconception about Machine Learning.

One of the biggest kept secrets in the Machine Learning communities is that it does not require a high understanding of statistics, mathematics, or any computer science degree to master.

Why do most believe that?

The inventions and paradigms used in Machine Learning were created by people with high-level degrees in these fields.

But let me ask you a question. Could you build a car from scratch? Probably not. Can you drive a car down to the grocery store? Probably yes.

You **don't** need to be a car mechanic to drive a car.

The same is true with Machine Learning—you do not need a high degree to use the models—you just need to know how to navigate them.

And the easiest and most powerful way to do that is with **Python**.

This explains why Python is the preferred language for Data Science and Machine Learning.

Read more here, get the projects, and see the videos.

Python for Finance: Technical Analysis

Learn **Python for Financial Data Analysis** with **pandas** (Python library) in this two-hour free eight-lessons online course.

The eight lessons will start you with **technical analysis** using **Python** and **Pandas**.

The eight lessons:

Copyright © 2023 learnpythonwithrune.org

- Get to know **Pandas** with Python to get historical stock price data
- Learn about **Series** from **Pandas** to make calculations with the data.
- Learn about **DataFrames** from **Pandas** to add, remove, and enrich the data.
- Start visualizing data with **Matplotlib** to learn how to understand price data.
- Learn to read data directly from **APIs** like **Yahoo! Finance** the right way.
- Calculate the **Volatility** and **Moving Average** of a stock.
- Technical indicators: **MACD** and **Stochastic Oscillator**, made easy with **Pandas**.
- Export it all into **Excel** in multiple sheets with color-formatted cells and charts.

Read more on the course page and see the video lectures.

The Code is available on **GitHub**.

Python for Finance: Risk and Return

Learn **Python for Finance** with Risk and Return with **Pandas** and **NumPy** (Python libraries) in this free 2.5-hour, eight-lesson online course.

The eight lessons will get you started with **technical analysis** for Risk and Return using **Python** with **Pandas** and **NumPy**.

The eight lessons:

- Introduction to **Pandas** and **NumPy** (i.e., Portfolios and Returns)
- **Risk** and **Volatility** of a stock – **Average True Range**
- **Risk** and **Return** – **Sharpe Ratio**
- **Monte Carlo Simulation** – Optimize portfolio with Risk and Return
- **Correlation** – How to balance portfolio with Correlation
- **Linear Regression** – how X causes Y
- **Beta** – a measure of a stock's volatility in relation to the overall market.
- **CAPM** – Relationship between systematic risk and expected return

Read more on the course page and see the video lectures.

The code is available on **GitHub**.

Copyright © 2023 learnpythonwithrune.org

A 21-hour course Python for Finance

Did you know that the number one killer of investment return is emotion?

Investors should not let fear or greed control their decisions.

How do you get your emotions out of your investment decisions?

A simple way is to perform objective financial analysis and automate it with Python!

Why?

- Performing financial analysis makes your decisions objective. You are not buying companies that your analysis did not support.
- Automating them with Python ensures that you do not compromise because you get tired of analyzing.
- Finally, it ensures that you get all the calculations done correctly.

Does this sound interesting?

- Do you want to learn how to use Python for financial analysis?
- Find stocks to invest in and evaluate whether they are underpriced or overvalued?
- Buy and sell at the right time?

This course will teach you how to use Python to automate the process of financial analysis on multiple companies simultaneously and evaluate how much they are worth (the intrinsic value).

You will get started in the financial investment world and use data science on financial data.

Read more on the course page.

Copyright © 2023 learnpythonwithrune.org