PYTHON Playground

Dive into Coding with 19 Fun and Educational Projects

Learn Python by Creating 19 Projects

Do you want to learn Python by creating projects?

In this ebook, you will learn Python in 19 projects with source code.

You will learn the following.

- **Benefits** of learning Python by creating projects
- Learn Python **quickly** by creating projects
- These projects will teach you **essential** programming skills
- How to get started with **installation** and Notebooks
- **19 projects** to teach you programming, with description and source code.
- Python using **libraries** and **frameworks**

Are you ready to get started in this practical course of 19 Python projects that will teach you Python?

Python is known for its simple and intuitive syntax, which makes it easy to read and write. This makes it an ideal language for beginners, as well as experienced programmers who want to quickly prototype new ideas.

Benefits of Learning Python by Creating Projects

There are numerous benefits to learning Python by creating projects.

- **Hands-on Learning**. By creating Python projects, you will gain hands-on experience with the language, which is one of the most effective ways to learn. You'll be able to see how Python works in practice and get a feel for the language's syntax and structure.

- **Practical Application**. When you create Python projects, you will have the opportunity to apply the concepts and techniques you've learned in a practical way. This can help you better understand how Python can be used to solve real-world problems.

- **Creativity**. Python projects offer a chance to flex your creative muscles and come up with unique solutions to problems. You can use Python to create anything from simple command-line utilities to complex web applications.

- **Improved Problem-Solving Skills**. As you work on Python projects, you will inevitably encounter challenges and problems that you'll need to overcome. This will help you develop your problem-solving skills and improve your ability to think critically.

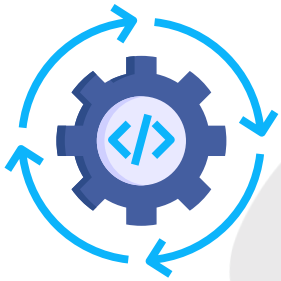
- **Portfolio Building**. Creating Python projects can be a great way to build up your portfolio and showcase your skills to potential employers or clients. This can help you stand out from other job candidates or freelancers.

Now you know why it is a good idea to learn Python by creating projects, next we need to know what you should learn.

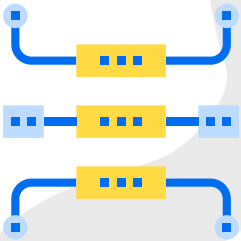
Learn Python Quickly by Creating Projects

To effectively learn Python quickly by creating projects, it is important to focus on the essential aspects of the language.

These fundamentals will provide a strong foundation for programming and prevent the distraction of unnecessary constructions that may not be useful.



- **Start with the basics**, including variables, data types, operators, and control structures. By mastering these foundational elements, you can solve simple problems and build a solid base for further learning.



- **Practice working with data structures**, such as lists, dictionaries, sets, and tuples. Data is essential to programming, and becoming proficient in Python's built-in data structures will help you manage and manipulate data effectively.

- **Learn how to use functions** to organize your code into smaller, reusable blocks. By solving larger problems in a structured way, you can test the functionality of each function and build more complex programs with ease.

- **Learn how to read and write files in Python.** This is a fundamental skill for many programming tasks, and understanding how to work with files is essential for more advanced applications.

- **Finally, become familiar with libraries**, as Python has a vast collection of them available to extend its functionality. Learn how to install and use popular libraries like NumPy, Pandas, and Matplotlib, which will enable you to create powerful applications with Python.

These Projects Will Teach You Essential Programming Skills

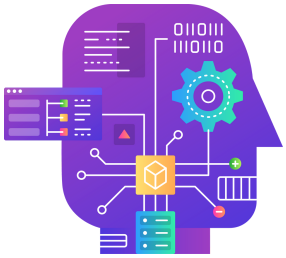
Learning Python by creating the following 19 projects will teach you the following skills.

- **String manipulation**. Building a program that can manipulate text strings will teach you how to work with character arrays, regular expressions, and data parsing. String manipulation is important because it is a fundamental skill in software development, and it is used in many programming tasks, such as data validation, text parsing, and cryptography.

- **User interaction**. Building a program that can interact with users will teach you how to handle user input, design user interfaces, and provide feedback to users. User interaction is important because it is essential in creating user-friendly software, and it helps developers understand the needs and expectations of their users.

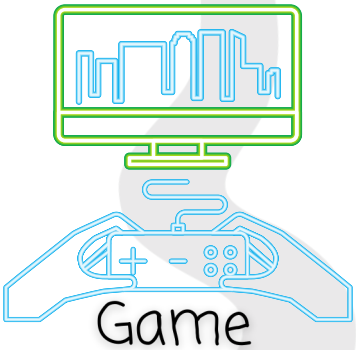
- **Data models**. Building a program that can handle data models will teach you how to design and implement data structures, work with databases, and manipulate data in memory. Data models are important because they provide a way to organize and structure data, making it easier to process, analyze, and store.

These Projects Will Teach You Essential Programming Skills



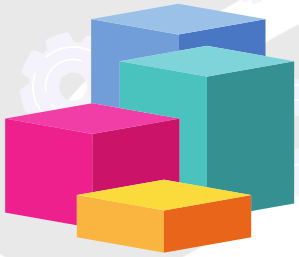
Algorithmic thinking

- **Algorithmic thinkin**g. Building a program that can implement algorithms will teach you how to analyze problems, design efficient solutions, and optimize code. Algorithmic thinking is important because it is a fundamental skill in computer science, and it is used in many programming tasks, such as sorting, searching, and optimization.



Game creation

- **Game creation**. Building a program that can create games will teach you how to use graphics, animations, and sound effects, and how to design engaging user experiences. Game creation is important because it is a fun and creative way to apply programming skills, and it can also teach you how to work in a team, manage deadlines, and iterate on designs.



Modular programming

- **Modular programming**. Building a program that can be broken down into smaller, reusable modules will teach you how to write clean, maintainable code, and how to collaborate with other developers. Modular programming is important because it makes it easier to debug, test, and extend code, and it also helps developers work more efficiently and avoid duplication of effort.

Data processing

- **Data processing**. Building a program that can process large amounts of data will teach you how to work with data streams, optimize code for performance, and use parallel processing techniques. Data processing is important because it is a common task in many fields, such as finance, healthcare, and science, and it can also help developers gain insights into complex systems and phenomena.

Hi, I'm Rune

I may not have learned **Python** quickly or possess any special talent in coding.

In fact, **I struggled** just like many others when I was starting out.

However, what sets me apart is my **passion for sharing** what I've learned and helping others succeed.

I take pride in being able to guide people with **no technical background** towards achieving their coding goals.

Through perseverance and a willingness to learn, I've been able to help others **succeed faster than me**.

The truth is, **coding can be easy**, and I'm passionate about sharing that message with others.

With over 20 years of experience and a Ph.D. in Computer Science, our instructor Rune has dedicated countless hours to creating a comprehensive resource for both beginners and advanced learners of Python.

How to get started with installation and Notebooks

Here's a short guide to get started.

Downloading and Installing Anaconda:

1. Go to the Anaconda website (sdf) and download the appropriate version of Anaconda for your operating system.
2. Follow the installation instructions for your operating system.
3. Once installed, launch the Anaconda Navigator.

Getting Notebooks from a GitHub Repository:

1. Go to the GitHub repository here.
2. Click on the "Code" button and select "Download ZIP" to download the repository as a ZIP file.
3. Extract the ZIP file to a folder on your computer.
4. Open the Anaconda Navigator and launch the Jupyter Notebook.

Launching the First Notebook:

1. In the Jupyter Notebook dashboard, navigate to the folder where you extracted the notebook files from the GitHub repository.
2. Click on the notebook file you want to open (it will have a ".ipynb" extension).
3. The notebook will open in a new tab in your web browser.
4. To run a cell in the notebook, click on the cell and press "Shift+Enter" or click on the "Run" button in the toolbar.

That's it! You should now be able to start your journey of 19 Python projects to learn Python.

19 Python Projects to Teach You Programming with Python

You will learn Python essentials by creating these 19 projects.

The 19 projects listed before we dive into them.

- **Project 00** Leet Speak
  - **String manipulation** to replace specific characters with others.
  - **Conditionals** and **loops** to apply the appropriate leet speak translations.
  - **Dictionaries** to store translations and avoid repetitive code.
  - Accepting **user input** and formatting output.

- **Project 01** Temperatur Converter
  - **User Input** and **Output** with Python's input() and print() functions.
  - **Variables** and **Data Types** including integers, floats, and strings.
  - **Basic Math Operations** such as addition, subtraction, multiplication, and division.
  - **Conversion Formulas** between Celsius and Fahrenheit temperature scales.
  - **Float Precision** and its impact on the user experience.

- **Project 02** Store
  - How to use **dictionaries** to track sales of unknown items.
  - Prompting for **user input**.
  - Converting string **input to integers**.
  - Setting **default values** in a dictionary.
  - **Iterating** over dictionary key-value pairs.

- **Project 03** ToDo List
  - **User interaction** techniques.
  - **Looping** until the user quits the program and iterating over a list.
  - Usage of Python's **list** data structure.
  - Implementing **conditionals** to allow user control.
  - **Enumerating** items in a list by their index.

19 Python Projects to Teach You Programming with Python

- **Project 04** ELIZA
  - The project allows for exploration and experimentation with **natural language processing** techniques.
  - It provides an opportunity to learn about **rule-based programming** and develop programming skills.
  - Creating ELIZA requires **analyzing** and **interpreting** user input, identifying keywords and patterns, and generating appropriate responses.
  - There are many online resources and examples available to guide **beginners** and **experienced** programmers alike.
  - ELIZA provides a fun way to **simulate human-like** conversation through programming.

- **Project 05** Fibonacci
  - The educational aspects of programming, such as **loops**, **conditionals**, and **recursion**.
  - The development of **efficient algorithms** using Fibonacci numbers.
  - **Mathematical modeling** of natural phenomena that utilize the Fibonacci sequence.
  - **Cryptography** algorithms that use Fibonacci numbers for encryption and decryption.

- **Project 06** Guess a Number Game
  - **Control flow:** Conditional statements and loops are needed to control the flow of the game based on user input and game logic.
  - I**nput and output** handling: User input must be handled, and output must be displayed to the user.
  - **Random number generation**: A random number must be generated, which can teach the programmer how to generate random numbers in their code.
  - **Debugging**: The game can help programmers practice debugging by identifying and fixing errors in their code.
  - **User interface design**: Designing a user-friendly interface for the game can teach the importance of user interface design in programming.

19 Python Projects to Teach You Programming with Python

- **Project 07** Maze Game
  - **Problem-solving**: breaking down complex problems into smaller tasks
  - Algorithmic thinking: creating algorithms to generate the maze
  - **User input and game logic**: handling user input and designing game logic
  - **User interface**: creating a simple user interface, and extending it to a graphical user interface using Pygame or Pyglet
  - **Skills development**: developing problem-solving skills, algorithmic thinking, user interface design, and game development skills.

- **Project 08** Heads or Tails
  - Heads-or-tails is a simple problem to solve in Python, but it can **teach you a lot**.
  - **Function structure**. Implementing heads-or-tails can help you learn how to structure your code in simple and organized functions.
  - **Randomness**. The problem requires generating a random result, which can teach you how to use Python's built-in random module.
  - **User input handling**. The problem also requires handling user input, which can teach you how to validate and prompt users for input in your programs.
  - **Loops**. To ensure that the user enters the correct input, the program needs to use loops to keep prompting the user until they enter valid input.
  - **Conditionals**. The program needs to validate the user's guess against the random result using conditional statements.

19 Python Projects to Teach You Programming with Python

- **Project 09** Basic Calculator
  - **Functions** are powerful tools that allow implementation in smaller blocks of code that can be tested individually.
  - Modularity is the main reason to use functions in Python programming.
  - **Modular programming** breaks down complex problems into smaller, more manageable pieces, making code easier to read, understand, and maintain.
  - Modular programming promotes code reuse, consistency, collaboration, and **program quality**.
  - **Breaking down a program** into smaller functions makes it easier to test each function individually, identify bugs and other issues more quickly, and ensure that the program is reliable, efficient, and easy to maintain.

- **Project 10** Interactive Board Game
  - Clarifies **project requirements**
  - **Structures** the project
  - **Saves** time
  - Facilitates **collaboration**

- **Project 11** Tic Tac Toe Game
  - **Creative expression**
  - **Problem-solving**
  - Learning **new technologies**
  - **Collaboration**
  - Building engaging **user interfaces**
  - Marketable **skills** for the tech industry

19 Python Projects to Teach You Programming with Python

- **Project 12** Data Processing
  - **Data processing** helps organizations make better decisions by analyzing, manipulating, and transforming large amounts of data.
  - Python is a **powerful language** for data processing due to its simplicity, flexibility, and extensive libraries.
  - Automating data processing tasks using Python can improve efficiency and reduce the time and effort required for manual data processing.
  - Python is easy to learn and use, making it an **ideal** language for data processing tasks, even for those without extensive programming experience.
  - Data processing can uncover **insights**, **patterns**, and **trends** that can inform future decisions for businesses and organizations.

- **Project 13** Test Processing
  - **Data analysis**: Python developers can use text processing techniques to extract valuable insights from large volumes of unstructured data as it is an important part of data analysis.
  - **Automation**: Python can be used to automate repetitive tasks such as cleaning and standardizing text data that involve text processing.
  - **Natural Language Processing (NLP)**: Python is a popular language for NLP due to its extensive libraries and tools for text processing. NLP is a rapidly growing field that involves analyzing and understanding natural language data.
  - **Machine learning**: Python has a wide range of machine learning libraries that are well-suited for working with text data, which is often used as input for machine learning models.

19 Python Projects to Teach You Programming with Python

- **Project 14** Acronym Generator
  - **String manipulation**: Working with strings is a core aspect of programming an acronym generator. A programmer will learn to extract and manipulate strings to create new words and acronyms.
  - **Loops**: An acronym generator may require the use of loops to iterate through strings and characters, such as when identifying the first letter of each word to form an acronym.
  - **User input**: The acronym generator is designed to take user input, and a programmer will learn how to handle and process input from users, including error handling.
  - **Functions**: An acronym generator can be broken down into smaller functions to make the code more modular and reusable. A programmer will learn how to create functions that perform specific tasks and can be used in other program parts.

- **Project 15** Password Generator
  - **String manipulation**: A password generator involves manipulating strings to create random sequences of characters. A programmer gains experience in string concatenation, slicing, and other operations commonly used in string manipulation.
  - **Random number generation**: Generating random numbers is a core component of any password generator. A programmer learns to use Python's built-in random module to generate random integers and other values.
  - **Conditional statements**: A password generator makes decisions based on user input, requiring the use of conditional statements (e.g., if, else) to handle different cases.
  - **Loops**: A password generator may repeat certain operations multiple times, requiring the use of loops (e.g., while, for) to iterate over a sequence of characters or numbers.
  - **Secure password generation**: A programmer learns about common password security best practices and how to implement them in code to create a secure password that cannot be easily guessed.

19 Python Projects to Teach You Programming with Python

- **Project 16** Anagram Game
  - **String manipulation**. Anagrams require manipulating strings to create new words or phrases. This teaches slicing, concatenation, and sorting techniques.
  - **Input and output**. Accepting user input and outputting results requires using Python's input() and print() functions.
  - **Looping**. The game requires iterating to prompt the user for guesses until they are correct.
  - **Data structures**. An efficient anagram game uses a data structure like a list or dictionary to store the dictionary of words for quick lookups.
  - **Randomness**. Using randomness is key to making the game fun. This teaches how to select random words and create random anagrams.

- **Project 17** Valid Parentheses
  - Python **syntax** and **semantics** for using conditional statements, loops, and string manipulation functions to check if parentheses are balanced or not.
  - **Problem-solving** skills for thinking through the logic of checking if parentheses are balanced and handling scenarios like nested parentheses.
  - **Algorithmic thinking** for breaking down the problem into smaller steps and designing a sequence of instructions to solve the problem.
  - **Debugging** skills for identifying and fixing errors in your code.
  - **Data structures** like stacks or queues to implement the parentheses checker algorithm and gain a deeper understanding of them.
  - **Code organization** by creating functions and modularizing your code for easy readability, maintenance, and reuse.

19 Python Projects to Teach You Programming with Python

- **Project 18** Four in a Row Game
  - **Logic and control flow**, including checking for win conditions and determining the best move using functions and conditional statements.
  - **Data structures** like lists and dictionaries manage and manipulate large amounts of game data efficiently.
  - **User interaction** with the game using a text-based interface.

Python in 12 weeks

From Zero to Python Hero in Just 12 Weeks: Mastering Libraries and Frameworks for Web Scraping, PDF Generation, Excel Automation, and More!

- This Python program is **designed for beginners** who want to learn the language from scratch and quickly become proficient in **using libraries and frameworks**.
- In just **12 weeks**, you will learn the basics of Python programming, including syntax, data types, control structures, and functions.
- You will then move on to more advanced topics, such as **object-oriented programming**, **file handling**, **web scraping**, and **data analysis**.
- We will teach you how to use popular Python libraries and frameworks, including **BeautifulSoup**, automate tasks, generate **PDFs**, and manipulate **Excel** files.
- This program includes **hands-on projects** and **real-world examples** that will help you apply your newfound knowledge and build a portfolio of work to showcase your skills.

By the end of the program, you will have the **confidence and expertise to use Python libraries and frameworks** to solve complex problems and build powerful applications.

Get 67% discount and join

Make a change in your life

- **Learning Python in 12 weeks** requires a commitment to consistent practice and a willingness to challenge yourself.

- By following the above guide, you can **acquire a solid foundation in Python programming** and gain hands-on experience with coding exercises and projects.

- Learning **Python is a valuable investment** in your personal and professional development, offering opportunities to build web applications, analyze and visualize data, and build machine learning models, among other applications.

- With **dedication and perseverance**, you can become proficient in Python and leverage its vast library ecosystem and community resources to build sophisticated applications.

- **Start your journey today** and in just 12 weeks, you'll be surprised at how much you've learned and how far you've come in your Python programming skills.

Rune

00 - Leet Speak

Implement a leet speak program

In this project to create a leet speak program with Python you will learn a lot.

- **String manipulation**. Leet speak involves converting certain letters to numbers or symbols. This will teach you how to manipulate strings in Python to replace specific characters with others.

- **Conditionals and loops**. Depending on the complexity of the leet speak program, you may need to use conditionals and loops to iterate over a string and apply the appropriate leet speak translations to each character.

- **Dictionaries**. To make the leet speak translation more efficient, you may use a dictionary to store the translations for each character. This can help avoid repetitive code and make the program more maintainable.

- **Input and output**. The program will need to take input from the user, in the form of a string of text, and output the corresponding leet speak translation. A Python developer can learn how to accept user input, and output the results in a formatted way.

Project Description

Leet (or “1337”), also known as eleet or leetspeak, is a system of modified spellings used primarily on the Internet. (wiki).

We take an input string from the user and convert it to a simplified leet that only converts a few letters. You can extend it further if you like.

This will create a leet speak program with Python and an example input-output pair.

Example
'TAKE ME TO ANOTHER LEVEL' → '74K3 M3 70 4N07H3R 13V31'

00 - Leet Speak

Project Breakdown

The first step in creating a project is to break it down into individual steps of what needs to be done.

This helps you as a programmer to keep it simple and not mix things together and create complex solutions. The most common mistake a programmer makes is to try to do too many things at the same time. Instead of just starting to program, you should try to think what is the logical flow that happens.

One solution could be as follows.

1. Input from the user.
2. Convert input to uppercase.
3. Make the leet conversion
4. Display the result.

This simple breakdown helps you have a simple flow in your program.

Step 1 Input from the user

The built-in function input() will prompt the user for an input and return it as a string.

```
phrase = input('Input phrase: ')
```

This will simply prompt the user and return the input from the user to the variable phrase.

00 - Leet Speak

Step 2 Convert input to uppercase

The string method upper() will return an uppercase version of the string.

```
phrase = phrase.upper()
```

If the user wrote ‘take me to another level’ then this will return ‘TAKE ME TO ANOTHER LEVEL’.

Step 3 Make the leet conversion

This is the core of the program and it is great to keep it isolated and not mixed up together with the other steps in the code.

A great way to transform a string into another string is to loop over it and generate it character by character.

The simplest way to make a character conversion is to use a dictionary for that.

```
lookup = {
    'A': '4',
    'E': '3',
    'I': '1',
    'L': '1',
    'O': '0',
    'S': '5',
    'T': '7'
}

leet = ''
for char in phrase:
    leet += lookup.get(char, char)
```

The lookup is a dictionary used to look up a character (the key) and another value (the value of the dictionary).

The loop iterates over each character in the string phrase and uses the dictionary get-method that will return the value if the key exists (the first argument) else it will use the default value (the second argument).

What it does if the character char is, say ‘L’, is to use lookup.get(‘L’, ‘L’) – in this case, it will return ‘1’ because it exists. On the other hand if lookup.get(‘K’, ‘K’) then it would return ‘K’because ‘K’ does not exist in the dictionary lookup.

Step 4 Display output

Finally, we need to display the result to the user. This can be done by using the print built-in function.

```
print(leet)
```

Now you have solved it.

00 - Leet Speak

The full code

Here you get the full code.

```
# 1: Input from the user
phrase = input('Input phrase: ')

# 2: Convert input to upper case
phrase = phrase.upper()

# 3: Make the leet conversion
lookup = {
 'A': '4',
 'E': '3',
 'I': '1',
 'L': '1',
 'O': '0',
 'S': '5',
 'T': '7'
}

leet = ''
for char in phrase:
 leet += lookup.get(char, char)


# 4: Display output
print(leet)
```

01 - Temperature Converter

Implement a temperature converter

Implementing a temperature converter in Python will teach you a lot of things.

It is a beginner project you need to have done at least once and it will teach you a lot.

- **User Input and Output**. You will learn how to accept user input and display output in the terminal using Python's built-in input() and print() functions.
- **Variables and Data Types**. You will learn how to use variables to store data and manipulate it using arithmetic operators. Additionally, you will learn about data types such as integers, floats, and strings.
- **Basic Math Operations**. You will learn how to perform basic math operations such as addition, subtraction, multiplication, and division in Python.
- **Conversion Formulas**. You will learn about the formula used to convert between Celsius and Fahrenheit temperature scales.
- **Float precision** You will learn how to work with float precision and how it matters for the user experience.

Are you ready?

01 - Temperature Converter

Project Description

Create a temperature converter in Python from Fahrenheit to Celsius using the formula

$°C=(°F-32)\times 5/9$

Project

- Prompt the user for temperature in Fahrenheit
- Calculate the temperature in Celsius
- Print the result

Example

- 75 → 23.89

Step 1 Prompt the user

You can use the built-in function **input()** to prompt the user. Notice that it returns a string, which needs to be converted to float for further calculations. That can be done with the built-in function **float()**.

```
# 1: prompt the user
fahrenheit_str = input('Input Fahrenheit: ')
fahrenheit = float(fahrenheit_str)
```

Step 2 Calculate the Celsius temperature

We will use the formula. Notice that you can use **round()** to get 2 digits (or any number of precision if you change the second argument).

°C=(°F−32)×5/9

```
# 2: Calculate the celsius temperature
celsius = (fahrenheit - 32)*5/9
celsius = round(celsius, 2)
```

02 - Store

Implement a Simple Store in Python

In this project, you will implement a simple store in Python.

This is a great project that will teach how things can be easily modeled with Python built-in data structures.

You will learn the following.

- **Dictionaries**. You will learn how to use Python dictionaries to keep track of what is sold, even though you don't know what items you sell. This will teach you the flexibility you have using the built-in data structures.
- **User input**. You will learn how to prompt the user for input.
- **Convert data types**. You will learn how to convert strings to integers in Python. This is something will encounter often when programming.
- **Default values**. You will learn how to use a dictionary and give a default value if the key does not exist already. This is by far one of my favorite things about Python dictionaries.
- **Iterate over the dictionary**. You will also learn how to iterate over the key-value pairs in a dictionary.

Now this is great. Are you ready?

02 - Store

Project Description

You will start to sell items from the awesome store you implement in Python in this project.

You count items sold, i.e. if you sell the three following items.

1. apple
2. pear
3. orange

To keep track of how many items you sell, you can use a dictionary.

The dictionary can use the items as keys, and the number of sold items as the value.

```
sold_items = {'apple': 0}
```

Create the following.

- Use a dictionary to keep track of sold items.
- Create a user prompt to sell items.
- Create a status view of items sold.

Step 1 The most important decision

When you need to solve a programming problem, the biggest decision you need to make is how to represent the data, the model, or whatever is the main thing in the project.

This is often not given in the project description.

In the above description, it is, but in most cases, it is not.

Therefore it is a good idea to get into the habit to investigate a few alternatives on how it can be represented.

Could you use a Python list instead?

02 - Store

Yes, you could keep a long list of all items sold during the day. Then at the end of the day, you could count the occurrences of each item in the list.

This would work too.

But it has some drawbacks.

- You keep the same item multiple times, and it does not give any value here.
- It will be more complicated to make a status of the sold items (you will see that in the end).

Hence we use a dictionary.

```
sold_items = {}
```

And surprisingly, we keep it empty.

Step 2 User prompt and update

Here you will see the power of using a dictionary.
But first, we need to prompt the user for what item and how many the user wants.

Afterward, we use the get method to update the dictionary.
Why, because get gives you the stored value or a given default value if the key does not exist.

```
item = input('What do you want to buy? ')
number_str = input('How many do you want? ')
number = int(number_str)

sold_items[item] = sold_items.get(item, 0) + number
```

Hence, the first time an item is looked up in the dictionary, it will return 0 and assign it to the number given. The second time, it will get the value and add number to it.

This actually fulfills the needs.

02 - Store

Step 3 Status of sold items

All you need is to print the key-value pairs from the dictionary.

```
for item, number in sold_items.items():
    print(item, number)
```

As you see, this step gets easy, because we used the dictionary as a data structure and not a list.

The full code

You have the full code here.

```
sold_items = {}

item = input('What do you want to buy? ')
number_str = input('How many do you want? ')
number = int(number_str)

sold_items[item] = sold_items.get(item, 0) + number

for item, number in sold_items.items():
    print(item, number)
```

02 - Store

Step 3 Status of sold items

All you need is to print the key-value pairs from the dictionary.

```
for item, number in sold_items.items():
    print(item, number)
```

As you see, this step gets easy, because we used the dictionary as a data structure and not a list.

The full code

You have the full code here.

```
sold_items = {}

item = input('What do you want to buy? ')
number_str = input('How many do you want? ')
number = int(number_str)

sold_items[item] = sold_items.get(item, 0) + number

for item, number in sold_items.items():
    print(item, number)
```

03 - ToDo List

Implement a ToDo list in Python

Implement a ToDo list in Python and you will learn a few things, including.

- **User interaction**. You will learn the nature of how to make a user interaction with Python.
- **Loop**. You will learn how to make a loop that will run until the user quits the program. Also, you will learn how to iterate over a Python list.
- **Data structures**. You will learn about Python's built-in data structure list, and see it can be used to create a ToDo list.
- **Conditionals**. You will learn to create conditionals, and how to make the program behave as the user want.
- **Enumerate**. You will learn how to enumerate the items in a list according to the index. This is a powerful skill to learn.

Are you ready for this?

Project Description

The ToDo list Python program you will write can do 4 things.

1. It can show the content of your ToDo list
2. It can add an item to your ToDo list
3. It can remove an item from your ToDo list
4. It can quit the program

03 - ToDo List

Step 1 The Data Model

Deciding how the data is represented is the most crucial decision. Most do not think that there are many options.

This choice is not given in the project description but impacts most of the other code. To see if a data model is a good choice, you should consider how it affects the implementation of the steps.

Here we will consider the Python list as our first choice.

To see if this is a good choice, we can see how it impacts the program features.

- It can show the content of your ToDo list – This can be done by iterating over the list and showing items.
- It can add an item to your ToDo list – This can be done by appending an item to the list.
- It can remove an item from your ToDo list – This can be done by removing an item from the list.
- It can quit the program – This is not relevant to the list.

You can consider other data structures or even implement your custom class.

Step 2 Program Structure

The overall structure of the program can be implemented in a while loop with conditionals. While the data structure we use is a Python list.

A great way to have it is in an infinite while loop, which will only be terminated if the user chooses the quit. option.

03 - ToDo List

Let's consider this code.

```
todo = []

while True:
    # print the options to the user
    print('1: Show content')
    print('2: Add an item to list')
    print('3: Remove an item from the list')
    print('4: Quit')
    # prompt the user
    option = input('Choose option: ')

    if option == '1':
        pass
    elif option == '2':
        pass
    elif option == '3':
        pass
    elif option == '4':
        print('Goodbye')
        break
    else:
        print('Not understood')
```

The quit option is implemented with a break in the infinite while loop (while True).

Now we just need to implement the rest of the code.

03 - ToDo List

Step 3 Implementation

The full code is implemented here.

```
todo = []

while True:
    # print the options to the user
    print('1: Show content')
    print('2: Add an item to list')
    print('3: Remove an item from the list')
    print('4: Quit')
    # prompt the user
    option = input('Choose option: ')

    if option == '1':
        for idx, item in enumerate(todo):
            print(idx, item)
    elif option == '2':
        item = input('Add item to list: ')
        todo.append(item)
    elif option == '3':
        idx_str = input('What item to remove (use
index): ')
        idx = int(idx_str)
        todo.pop(idx)
    elif option == '4':
        print('Goodbye')
        break
    else:
        print('Not understood')
```

We use an enumerate under option 1 to get the index of each item in the list. This makes our implementation easier, as you can use this index when you delete an item under option 2.

04 - ELIZA

What is ELIZA?

To understand the Python project on ELIZA, we need a bit of background.

ELIZA is a computer program designed to simulate human-like conversation by using natural language processing techniques. It was created by Joseph Weizenbaum in 1966 at the Massachusetts Institute of Technology (MIT) and is considered one of the earliest examples of a chatbot.

The program was named after the character Eliza Doolittle from the play “Pygmalion,” who undergoes a transformation from a working-class girl to a refined lady through language training.

ELIZA works by analyzing the user’s input, searching for keywords, and using pre-programmed rules to generate responses that mimic those of a therapist. It was designed to simulate a Rogerian psychotherapist, a type of therapist who emphasizes empathy and reflection rather than offering direct advice or solutions.

ELIZA became famous in the late 1960s and early 1970s, as it was one of the first programs to create the illusion of a human-like conversation. It has since inspired numerous chatbots and conversational agents, and its legacy can still be seen in the development of modern AI-powered chatbots and virtual assistants.

Implement ELIZA in Python

ELIZA can be a fun programming project to make because it allows you to explore and experiment with natural language processing techniques and simulate human-like conversation. The project can also be a great opportunity to learn about rule-based programming and develop your programming skills.

04 - ELIZA

Creating an ELIZA program can be challenging, as it requires analyzing and interpreting user input, identifying keywords and patterns, and generating appropriate responses. However, it can also be rewarding to see your program engage in convincing conversations with users.

Moreover, because ELIZA is a well-known and widely studied program, there are many resources and examples available online to help guide you through the process of creating your own version of the program. This can make the project more accessible and enjoyable for beginners and experienced programmers alike.

Overall, ELIZA can be a fun programming project to make because it allows you to explore the fascinating world of natural language processing while creating a program that can simulate a human-like conversation.

How ELIZA works?

It looks for simple patterns and substitutes to give the illusion of understanding from the computer (wiki).

Example

- You write I need cake it can look for a pattern I need *
- Then it can ask Why do you need cake?

04 - ELIZA

This can be done simply as follows.

```
eliza_response = 'What do you need?'

while True:
    s = input(eliza_response + ' ')

    if 'I need ' in s:
        your_need = s.split('I need ')[-1]
        eliza_response = 'Why do you need ' + your_need + '?'
    elif 'because' in s:
        your_cause = s.split()[-1]
        eliza_response = 'Why are you ' + your_cause + '?'


    elif s == 'quit':
        break
    else:
        eliza_response = 'Can you tell me why?'
```

This can generate the following dialog.

```
What do you need? I need cake
Why do you need cake? because I am hungry
Why are you hungry? because it makes me less irritated
Why are you irritated? because I need to avoid low
blood sugar
Why do you need to avoid low blood sugar? because I
need a reward
Why do you need a reward? because I was good
Why are you good? because I need to learn Python
Why do you need to learn Python? because I need a job
Why do you need a job? quit
```

You can expand on this idea to make the conversation more real

05 - Fibonacci

Implement Fibonacci in Python

Implementing Fibonacci in Python is one of the classical problems you need to master. It can be done in many ways, and we will cover the two main ones here.

Don’t let that stop you to proceed.

Mastering the Fibonacci problem in Python can teach you a lot.

- **Educational purposes**. Fibonacci is a classic mathematical sequence that is often used as a teaching tool for introductory programming courses. By programming the Fibonacci sequence, students can learn about loops, conditional statements, and recursion.

- **Algorithm development**. Fibonacci numbers are used in various algorithms, such as dynamic programming, graph algorithms, and search algorithms. By knowing how to generate Fibonacci numbers, programmers can implement these algorithms more efficiently.

- **Mathematical modeling**. The Fibonacci sequence appears in various natural phenomena, such as the branching of trees and the arrangement of leaves on a stem. By programming the Fibonacci sequence, programmers can model and simulate these phenomena in their programs.

- **Cryptography**. Fibonacci numbers are also used in some encryption and decryption algorithms. By programming the Fibonacci sequence, programmers can develop or implement cryptographic algorithms that rely on these numbers.

Are you ready for that?

05 - Fibonacci

Project Description

The Fibonacci sequence is as follows.

0 1 1 2 3 5 8 13 21 34 ... (continues)

It is generated as follows.

The next number is generated by adding the two last numbers.

0 + 1 = 1
1 + 1 = 2
1 + 2 = 3
2 + 3 = 5
3 + 5 = 8
8 + 13 = 21
13 + 21 = 34
21 + 34 = 55
...

Write a program that prints the Fibonacci sequence.
You will do it in two ways in this project.

Step 1 The direct way

How to solve this?

Well, the first thought is to make it in an iterative way.

```
def fib(n):
    if n <= 1:
        return n

    i = 0
    j = 1

    for _ in range(n):
        i, j = j, i + j

    return i
```

05 - Fibonacci

You try to follow how the sequence is calculated and this can be done but seems easy to get lost in the code.

To generate the sequence you can run this code.

```
for i in range(10):
 print(fib(i))
```

Step 2 The simple way (recursive)

The first time you see recursion, it might not be easy to understand.

But after inspecting the following code, you might find it easier to write and follow.

```
def fib(n):
    if n <= 1:
        return n

  return fib(n - 1) + fib(n - 2)
```

This actually generates the same sequence.
The code is actually writing the exact formula for generating it.
You can get the sequence as follows.

```
for i in range(10):
    print(fib(i))
```

06 - Guess a Number Game

Implement Guess a Number Game in Python

If you implement a guess-a-number game in Python you will learn many aspects of programming.

- **Control flow.** The game requires the use of conditional statements and loops to control the flow of the game based on user input and game logic.
- **Input and output handling**. The game requires the handling of user input and displaying output to the user.
- **Random number generation**. The game requires generating a random number that the user has to guess, which can teach the programmer how to generate random numbers in their code.
- **Debugging**. Debugging is an important skill in programming, and implementing a guess-a-number game can help programmers practice debugging by identifying and fixing errors in their code.
- **User interface design**. While a simple game like guess-a-number may not require a complex user interface, it can still teach the programmer the importance of designing a user-friendly interface for their programs.
- **Code organization**. The game requires the organization of code into functions and modules, which can help the programmer learn how to write modular and reusable code.

Are you ready?

06 - Guess a Number Game

Project Description

In this tutorial we will create the classical game: Guess a Number. The game can be described as follows.

- The computer generates a random number from 1 to 100.
- The user should guess it.
- If the user guesses correctly, it should print how many guesses the user used and the end.
- If the user guesses too low, print it.
- If the user guesses too big, print it.

Step 1 Get a random number

Python has a standard library **random**.

An easy way to generate random numbers is by using **randrange**. From the docs:

*random.**randrange**(stop)*
*random.**randrange**(start, stop[, step])*
Return a randomly selected element from range(start, stop, step).

If you are familiar with **range**, this is straightforward to use. We need a random number from the range of 1 to 100 (both inclusive). Two ways to get that are as follows.

```
from random import randrange
# will import randrange

# Gives a random integer from the range 1 to 100
# (both inclusive)
random_number = randrange(100) + 1

# Alternatively
random_number = randrange(1, 101)
```

I prefer the first way of doing it.

06 - Guess a Number Game

Step 2 User interaction and checks

Now you need user interaction to create the guessing game.
A thing to notice is, that you don't know how many times the user will use to guess the number.

One way to make an iterative approach is to use a while-loop and conditionals.

Another thing we need is to make sure that the input from the user is in the correct range.

```
from random import randrange

random_number = randrange(100) + 1

print("I'm thinking of a number between 1 and 100 (both
inclusive)")

while True:
    guess_str = input('What is your guess (1-100): ')
    guess = int(guess_str)

    if guess < 1:
        print('Out of bound')
        continue

    if guess > 100:
        print('Out of bound')
        continue
```

WARNING: the above code will continue forever, but we are not done with the project.

What to notice is, we use while True which generates an infinite loop.

We first use continue in the loop, if the user input is not in the expected bound. What continue does, is, it jumps up to the top of the loop and starts all over.

06 - Guess a Number Game

Step 3 The game

The full implementation of the game could be done as follows.

```
from random import randrange

random_number = randrange(100) + 1

guesses = 0

print("I'm thinking of a number between 1 and 100 (both
inclusive)")

while True:
    guess_str = input('What is your guess (1-100): ')
    guess = int(guess_str)

    if guess < 1:
        print('Out of bound')
        continue

    if guess > 100:
        print('Out of bound')
        continue

    guesses += 1

    if guess == random_number:
        print('Awesome! You guesses it!')
        print('You used', guesses, 'guesses')
        break

    elif guess < random_number:
        print('Too low')
    elif guess > random_number:
        print('Too high')
```

Notice that we use **break** to break out of the loop if the user guesses correctly.

07 - Maze Game

Implement a Maze Game in Python

Implementing a maze game in Python can teach you a lot.

- **Problem-solving**. Implementing a maze game requires breaking down a complex problem into smaller, more manageable tasks. This process involves identifying the rules of the game, designing the maze, creating the game logic, and handling user input. Problem-solving is an essential skill for any programmer, and implementing a maze game provides a great opportunity to develop it.

- **Algorithmic thinking**. To create a maze game, you need to generate a maze that the player can navigate through. This task requires algorithmic thinking, which involves breaking down a complex problem into a series of simple steps that can be automated. Implementing a maze game provides an opportunity to practice algorithmic thinking and develop problem-solving skills.

- **User input and game logic**. Implementing a maze game requires handling user input and designing the game logic. This involves creating an event loop that waits for user input, processing user input, updating the game state, and displaying the game screen. Implementing a maze game can help you learn how to handle user input and develop game logic.

- **User interface**. Implementing a maze game can help you learn how to create a user interface. In this implementation, it will be simple, but you can extend it to a graphical user interface (GUI) by using Python libraries like Pygame or Pyglet. You can learn how to draw images, display text, and respond to user input using a GUI. This can help you develop skills in creating user-friendly applications with good user interfaces.

Are you ready?

07 - Maze Game

Project Description

You will create a maze game in Python.

In the end, you will have an interactive game, where you can create the most amazing and complex mazes on your own.

Step 1 Representing the maze

To keep the flexibility of the game as big as possible, you want a great way to represent a maze of your own choice.
Here I present a simple maze, but it is up to you to make your own complex maze afterward.

```
maze = [
    list('########################'),
    list('#### ###  #  # ### #'),
    list('# #  ## # # #    # '),
    list('# # #   #  ##### # #'),
    list('# # # ########  # # #'),
    list('# # #     #  # # # #'),
    list('#  # ## #  #  #  #'),
    list('########################'),
]
```

The walls in the maze are represented by **#** and you place the player somewhere in the maze.

The maze is represented by a list of lists. Each list represents a row of items of either wall (#) or space ( ).

To make it simple, we will keep a player's location using the same coordinate system as the list of lists **maze** has.

Hence, a location like **x = 2** and **y = 1** is given by the 3rd row (indexed 2) and 2nd column (indexed 1).

07 - Maze Game

Step 2 Displaying the view of the player

The player will only see the nearby view of the maze. Meaning it will see if there is a wall over, under, left, and right of the location where it is.

An easy way to display that could be as follows.

```
maze = [
    list('########################'),
    list('#### ###  #  # ### #'),
    list('# #  ## # # #    # '),
    list('# # #   #  ##### # #'),
    list('# # # ########  # # #'),
    list('# # #     #  # # # #'),
    list('#  # ## #  #  #  #'),
    list('########################'),
]

x = 2
y = 1

# This will display the player view
print('###' + maze[x-1][y]*3 + '###')
print('###' + maze[x-1][y]*3 + '###')
print(maze[x][y-1]*3 + ' o ' + maze[x][y+1]*3)
print('###' + maze[x+1][y]*3 + '###')
print('###' + maze[x+1][y]*3 + '###')
```

Obviously, you are free to make it more advanced.

But this will look the location above (x – 1, y), left (x, y – 1), right (x, y + 1), and below (x + 1, y).

If you change the player's location, you can see it will update the player view accordingly.

07 - Maze Game

Step 3 Player interaction

To get player interaction we need input from the player.

```
from IPython.display import clear_output

maze = [
    list('######################'),
    list('#### ###  #  # ### #'),
    list('# #  ## # # #    # '),
    list('# # #   #  ##### # #'),
    list('# # # ########  # # #'),
    list('# # #     #  # # # #'),
    list('#  # ## #  #  #  #'),
    list('######################'),
]

x = 2
y = 1

while True:
    print('###' + maze[x-1][y]*3 + '###')
    print('###' + maze[x-1][y]*3 + '###')
    print(maze[x][y-1]*3 + ' o ' + maze[x][y+1]*3)
    print('###' + maze[x+1][y]*3 + '###')
    print('###' + maze[x+1][y]*3 + '###')

    move = input('w: up, s: down, a: left, d:right (h: map) - ')
    clear_output()

    if move == 'w': # up
        pass
    elif move == 's': # down
        pass
    elif move == 'a': # left
        pass
    elif move == 'd': # right
        pass
    elif move == 'h': # show map
        pass
    elif move == 'q': # quit
        break
```

For now, we have not implemented the controls except the quit. Notice we use a **while True** loop, which is an infinite loop, and we exit the loop by using the **break** in the quit option **q**.

As you notice, we start by displaying the player's view. Then we prompt for input and after that we clear the output (clear_output()), which will clear the output for text, to avoid rows of printing output from the last views. The clear_output() is specific for Jupyter Notebooks.

Step 4 Implementing show map

To help the player navigate in the maze, we can show the map if the player presses **h**.

```
# excluded code

while True:
    print('###' + maze[x-1][y]*3 + '###')
    print('###' + maze[x-1][y]*3 + '###')
    print(maze[x][y-1]*3 + ' o ' + maze[x][y+1]*3)
    print('###' + maze[x+1][y]*3 + '###')
    print('###' + maze[x+1][y]*3 + '###')

    move = input('w: up, s: down, a: left, d:right (h: map) - ')
    clear_output()

    if move == 'w': # up
        pass
    elif move == 's': # down
        pass
    elif move == 'a': # left
        pass
    elif move == 'd': # right
        pass
    elif move == 'h': # show map
        maze[x][y] = 'o'
        for row in maze:
            print(''.join(row))
        maze[x][y] = ' '
        input('Press enter to continue')
        clear_output()
    elif move == 'q': # quit
        break
```

As you can see, we start by inserting the player as an o and then we print each row of the maze by joining all the items. This will display the map of the maze.

Also notice, that after we have displayed the map, we change the player position to space again.
Finally, we prompt the user to press enter to continue and remove the map.

Step 5 Implementing the navigation

Finally, we need to implement the navigation.

```
# The following import is specific for JuPyter Notebooks.
from IPython.display import clear_output

maze = [
    list('######################'),
    list('#### ###  #  # ### #'),
    list('# #  ## # # #    # '),
    list('# # #   #  ##### # #'),
    list('# # # ########  # # #'),
    list('# # #     #  # # # #'),
    list('#  # ## #  #  #  #'),
    list('######################'),
]

x = 2
y = 1

while True:
    print('###' + maze[x-1][y]*3 + '###')
    print('###' + maze[x-1][y]*3 + '###')
    print(maze[x][y-1]*3 + ' o ' + maze[x][y+1]*3)
    print('###' + maze[x+1][y]*3 + '###')
    print('###' + maze[x+1][y]*3 + '###')

    move = input('w: up, s: down, a: left, d:right (h: map) - ')
    clear_output()

    if move == 'w': # up
        if maze[x - 1][y] == '#':
            print('Illegal move')
        else:
            x -= 1
    elif move == 's': # down
        if maze[x + 1][y] == '#':
            print('Illegal move')
        else:
            x += 1
```

```
elif move == 'a': # left
    if maze[x][y - 1] == '#':
        print('Illegal move')
    else:
        y -= 1
elif move == 'd': # right
    if maze[x][y + 1] == '#':
        print('Illegal move')
    else:
        y += 1
elif move == 'h': # show map
    maze[x][y] = 'o'
    for row in maze:
        print(''.join(row))
    maze[x][y] = ' '
    input('Press enter to continue')
    clear_output()
elif move == 'q': # quit
    break
else:
    print('Unknown move')

if y == len(maze[0]) - 1:
    print('You finished')
    break
```

It is important to check that the move is legal. This is done by checking if the move will result in a wall brick (#) or not.

Finally, we have kept the maze simple and only have one exit on the far right. This way, a simple way to check if the player has finished the maze is to see if the move results in a position on the far right side of the maze.

What you can do next?

Well, I am glad you asked.

You can create your own maze now. If you keep the only exit in your maze at the far right, keep the maze rectangular, then you can use the code as it is.

If you make further changes, you might need to modify the code to your specific needs.

08 - Heads or Tails

Implement Heads or Tails in Python

While heads-or-tails is a simple problem to solve in Python you can learn from it.

- **Simplify**. How to structure your code in a simple way in functions. This is one of the skills you need a lot of practice in, while it seems simple and easy, there is a lot more to it.
- **Randomness**. You need to flip a coin and for that, you need randomness.
- **User prompt**. When you program user input can be in the wrong format. How do you deal with that?
- **Loops**. To make a user prompt require correct input, you need to understand loops and how to use them in this case.
- **Conditional**. You need to validate the guess from the user with the random coin. To do that you need conditionals.

Are you ready?

Project Description

This is a simple project to create a heads-or-tails game in Python, we need to learn to work with functions.

You should build a guessing game of heads or tails.

Game description.

1. The user is asked to guess head or tails
2. The game will “flip” a coin to either heads or tails.
3. The game will write if the user guessed correctly or not.

Your goal is not to write the code, it is to use functions to create it. Note: An advantage of writing functions is that it enables you to test it isolated.

08 - Heads or Tails

Step 1 Understanding how to break it down

Why bother breaking the problem down into isolated blocks?

To simplify the code.

Instead of “just starting to write the code” it is always a good idea if you can break the problem down in smaller pieces that can be treated isolated.

The power is it will make it easier to implement isolated pieces and the best part is, you can test pieces of code isolated.
Let’s try to do it for this project.

One way to simplify is to think about what is happening in the game.

1. The user is asked to guess heads or tails
2. The game will “flip” a coin to either heads or tails.
3. The game will write if the user guessed correctly or not.

That could actually be one simple way to break it down into the above 3 pieces.

Now your job is to implement them isolated and test that they work as expected.

Step 2 Prompt the user

A great way to organize code is to implement it in functions.

This organizes the code in blocks of code that you can use with a call to a function. But more importantly, you can test it isolated.

One way to prompt the user could be as follows.

08 - Heads or Tails

```
def input_from_user():
    while True:
        guess = input('head or tails? ')

        if guess in ['head', 'tails']:
            return guess

        print('Not valid input!')
```

This uses the infinite loop (**while True**) to prompt the user (**input()**), until the input is valid (**if guess in...**).

Now you can test that function isolated to see if it does what is expected.

Step 3 Flip a coin

The next piece we need to implement is a flip of a coin.

To do that we need randomness. Luckily, **Python** has a standard library **random** that can assist you.

We will use **randrange** as it is a standard go-to function to use to get a random integer.

One way to implement it could be as follows.

```
from random import randrange

def flips_a_coin():
    coin = randrange(2)
    if coin == 0:
        return 'tails'
    else:
        return 'head'
```

The call **randrange(2)** will either return 0 or 1.

If it returns 0 we return **tails** otherwise **head**.

08 - Heads or Tails

Step 4 Print the result

Finally, we need to validate if the user's guess is correct.

This is where the power of functions is great.

```
def print_result(user_guess, coin):
    if user_guess == coin:
        print('You guessed it! You won!')
    else:
        print('No! You guessed wrong!')
```

You might wonder why I am excited about this function.

Well, this is where you actually combine the output of the previous two functions.

But here is the great part.

You can test it isolated.

You can try all possibilities manually with your function.

```
print_result('tails', 'tails')
print_result('head', 'tails')
print_result('tails', 'head')
print_result('head', 'head')
```

Now you know that all the pieces work isolated.

It is time to combine it all.

08 - Heads or Tails

Step 5 Combining it all

We know that all functions work isolated.

Now it is time to combine it all.

```
user_guess = input_from_user()

coin = flips_a_coin()

print_result(user_guess, coin)
```

I must admit. This is beautiful, it is simple.

Why is this powerful?

Because it is simple to understand. If you notice something is wrong, say, if it only flips tails, it is easy to identify where in the code you should look, and on top of that, you can test that piece of code isolated.

09 - Basic Calculator

Modularity Why it is important

In this project on a basic calculator in Python, you will learn how to break things down into functions and why you should do that.

Functions give you the power to implement things in smaller blocks of code, which you can test individually.

The main reason to use Python functions is Modularity.

Modularity Functions help to break down complex programs into smaller, more manageable pieces. This makes the code easier to read, understand, and maintain.

Modular programming is an important skill to master because it helps programmers to break down complex problems into smaller, more manageable pieces. By breaking down a program into smaller functions, it becomes easier to understand, test, and modify.

Modular programming promotes code reuse, which means that functions can be written once and used in multiple places throughout a program, or even in different programs altogether. This reduces the amount of code that needs to be written and maintained and also helps to ensure consistency across different parts of a program.

In addition, modular programming makes it easier to collaborate with other programmers. By breaking down a program into smaller functions, different programmers can work on different parts of the program without interfering with each other's work. This can make it easier to develop large, complex programs in a team environment.

Finally, modular programming can help to improve the overall quality of a program. By breaking a program into smaller functions, it becomes easier to test each function individually, which can help to identify bugs and other issues more quickly. This can help to ensure that a program is reliable, efficient, and easy to maintain.

09 - Basic Calculator

Project Description

In this project, you will create a simple calculator.
The calculator will take input from the user and create the calculation.

It will only accept positive integer input with one operator.

Examples of valid input.

- 1+4
- 6*10
- 100-90
- 1000/50

Examples of invalid input.

- 1+2+3
- 1/3*6
- -10/3
- 10*-1

Step 1 Design Choices

When a developer gets a task, often there are things not specified.

Examples
- What happens if the user inputs data in the wrong format (invalid input)?
- How should it output the result data?

What to do?
- Sometimes you can clarify these issues with the user.
- Other times you can make choices based on your knowledge or best guesses.

When to do what?
- Who is the user or owner of the code you develop?
- How big an impact does the decision have?

09 - Basic Calculator

Step 2 Breaking it down

A great way to break projects down is to divide them into the normal flow you would think of it.

1. Input from the user and validation of input
2. Calculate the result
3. Output the result

Then each of the above 3 steps can be divided further down to make the code more simple.

Step 3 Input from the user

The first thing to realize is that the validation is difficult to implement and needs proper testing to make sure it is done correctly.

One thing you can do before that is to make sure the input functionality works independently of the validation.

def is_input_format_correct(input_str):
  return False

```
# Input from user and validation of input

def input_calculation():
    """
    Prompts the user for a calculation.
    Repeat until input from user is on format [num][op][num]
    Return the input from user.
    """
    while True:
        input_str = input('Input calculation: ')

        if is_input_format_correct(input_str):
            return input_str

        print('Expected format: [number][operator][number]')
```

09 - Basic Calculator

We first have the validation function, which returns False by default. This function will be implemented afterward. This enables you to test whether the input calculation function works as expected.

The input_calculation() uses an infinite while-loop and breaks out of it by using the return statement from a function.

Step 4 Breaking the validation down

The function is_input_format_correct(input_str) needs to validate 3 things.

1. If input_str only contains the following characters 0123456789+-*/
2. If input_str only contains one operator (one of the following +, -, *, or /).
3. If input_str is on format [number][operator][number] (example 123+456)

A great way to do that is to make 3 functions for that. This way you can test if they do what you expect.

We keep the functions simple and do not use any advanced Python. But it uses for-loops and conditional statements.

```
def valid_chars(input_str):
    """
    Returns True if input_str only contains chars from
'0123456789+-*/'
    """
    for c in input_str:
        if c not in '0123456789+-*/':
            return False
    return True

assert valid_chars('123+456')
assert valid_chars('0123456789+-*/')
assert valid_chars('123b123') == False
```

09 - Basic Calculator

If you are unfamiliar with assert it evaluates the expression followed and raises an exception if it evaluates to anything else than True.

That is a great way to test if your function works as expected.

```
def one_operator(input_str):
    """
    Given input_str only contains chars '0123456789+-*/'
    Return True if input_str only has one operator (+, -, *,
or /)
    """
    no_add = input_str.count('+')
    no_minus = input_str.count('-')
    no_mult = input_str.count('*')
    no_div = input_str.count('/')

    no_operators = no_add + no_minus + no_mult + no_div

    if no_operators == 1:
        return True
    else:
        return False

assert one_operator('123+123')
assert one_operator('123-123')
assert one_operator('123*123')
assert one_operator('123/123')
assert one_operator('123123/')
assert one_operator('123++123') == False
assert one_operator('123+/123') == False
assert one_operator('123+*123') == False
assert one_operator('123--123') == False
```

The power of writing a doc string here, is, that you can give assumptions to the user. The function uses the string method count. This function will only work correctly if the input is of a certain format (a format passing the previously defined function).

Now that is powerful.

09 - Basic Calculator

```
def correct_format(input_str):
    """
    Given input_str only contains chars '0123456789+-*/'
    and input_str only has one operator (+, -, *, or /)
    Return True if input_str is on the format [num][op][num]
    """
    if input_str[0] in '+-*/':
        return False

    if input_str[-1] in '+-*/':
        return False

    return True

assert correct_format('0+0')
assert correct_format('1+1')
assert correct_format('2+2')
assert correct_format('3+3')
assert correct_format('4+4')
assert correct_format('5+5')
assert correct_format('6+6')
assert correct_format('7+7')
assert correct_format('8+8')
assert correct_format('9+9')
assert correct_format('99+') == False
assert correct_format('+99') == False
assert correct_format('99-') == False
assert correct_format('-99') == False
assert correct_format('99*') == False
assert correct_format('*99') == False
assert correct_format('99/') == False
assert correct_format('/99') == False
```

I think I “over”-asserted the above function.

Anyhow. We have now higher confidence that all 3 functions work as expected.

Now we can combine them.

```
def is_input_format_correct(input_str):
    """
    Return True if input_str is on the format [num][op][num]
    """
    if not valid_chars(input_str):
        return False

    if not one_operator(input_str):
        return False

    if not correct_format(input_str):
        return False

    return True

assert is_input_format_correct('123+123')
assert is_input_format_correct('123b123') == False
assert is_input_format_correct('123++123') == False
assert is_input_format_correct('123123+') == False
```

Now we have implemented the validation and we can use it together with the input_calculation().

Step 5 Calculate the result

Now the power of having the validation of the format before this step, is, that you now can assume the input to this function is of the correct format.

09 - Basic Calculator

```
def convert_to_ints(numbers_str):
    """
    Input is a list of numbers as str.
    Returns a list of the numbers as int.
    """
    numbers = []

    for number_str in numbers_str:
        numbers.append(int(number_str))

    return numbers

assert convert_to_ints(['123', '123']) == [123, 123]


def calculate_result(calc):
    """
    If calc is on format [num][op][num]
    Returns the result of the calculation calc.
    """
    if '+' in calc:
        numbers_str = calc.split('+')
        numbers = convert_to_ints(numbers_str)
        return numbers[0] + numbers[1]
    if '-' in calc:
        numbers_str = calc.split('-')
        numbers = convert_to_ints(numbers_str)
        return numbers[0] - numbers[1]
    if '*' in calc:
        numbers_str = calc.split('*')
        numbers = convert_to_ints(numbers_str)
        return numbers[0]*numbers[1]
    if '/' in calc:
        numbers_str = calc.split('/')
        numbers = convert_to_ints(numbers_str)
        return numbers[0]/numbers[1]

assert calculate_result('123+1') == 124
assert calculate_result('123-1') == 122
assert calculate_result('123*1') == 123
assert calculate_result('1/2') == 0.5
```

The function only expects and should only calculate correctly if the calculation string is in the correct format.

Also, we created a simple helper function. I know it can be done simply with list comprehension(more here), but we are making the code easy to understand and easy for everybody to enjoy.

Step 6 Display the result

The last piece is pretty straightforward.

```
def display_result(calc, result):
    print('The calculation of', calc)
    print('is', result)
```

Feel free to make it nicer than I have the capability to do.
The power of breaking it down as we did, is, that it makes it easy to improve or change each piece isolated.

Step 7 Combining it all

Now, we need to combine it all.

```
# Combine it all
calc = input_calculation()

result = calculate_result(calc)

display_result(calc, result)
```

I hope you see the power of that.

3 lines of code and you have it all there. If you know something is wrong with the way you calculate it, you know where to look.

Also, if you want to make a more advanced calculator you know how to proceed now.

10 - Interactive Board Game

Project Description

To implement an interactive board game in Python you can consider the following board.

```
-------------
|o|  |  |  |x|
-------------
```

The player is O and the goal is to not get hit by X.

The player can move either 1 or 2 moves to left or right, but only inside the board.

The computer will make a similar move with X.

If the player lands on X or if the computer lands on O the player loses.

Importance of Design

Making a design of your Python project before starting to code is crucial for several reasons, especially when you make an interactive board game in Python.

- **Clarifying the project requirements**. Creating a design document helps to identify the requirements of the project, the data that needs to be collected or generated, and the expected outputs. This can prevent misunderstandings and ensure that everyone involved in the project is on the same page.

- **Structuring the project**. Designing your project helps you break it down into manageable parts and plan out how those parts will interact. This can help you identify potential issues early on and make sure that the project is structured in a way that will be easy to maintain and expand.

10 - Interactive Board Game

- **Saving time**. A well-designed project can save a lot of time in the long run by reducing the number of errors and simplifying the debugging process. By planning ahead, you can ensure that the project is built in a way that is efficient, maintainable, and scalable.

- **Facilitating collaboration**. When working on a project with others, a design document can help ensure that everyone has a clear understanding of what is being built and how it will work. This can facilitate collaboration and help avoid miscommunications.

A great way to break things down is to think about the logical steps in the process of the game.

One way to do that is as follows.

1. Represent the board
2. Display the board
3. Input from the user (with validation)
4. Check if the user lost
5. Update the board
6. Let computer move
7. Check if the computer won
8. Update board for computer

The next part is to implement each part of the steps and finally combine them. A great way to do that is to keep each step in one or more functions.

Step 0 Represent the board

While this seems to be a simple decision, it often has the biggest impact on the code. Said differently, the code of the interactive board game in Python will be influenced by this decision.

How you represent the board will have an impact on most other parts of your implementation.

10 - Interactive Board Game

A simple way to do it is as follows, but still, consider that it will have an impact on the rest of the code you need to write.

```
board = ['O', ' ', ' ', ' ', ' ', 'X']
```

We have it as a list of characters using the symbols O and X and space for empty.

Step 1 Display the board

This can be done in many ways, but here we just keep it simple displaying output with print statements.

```
# Please Notice that this is only used in Notebooks.
from IPython.display import clear_output

def display_board(board):
    clear_output()
    length = len(board)

    print('-'*(length*2 + 1))
    for item in board:
        print('|', item, sep='', end='')

    print('|')
    print('-'*(length*2 + 1))
```

As mentioned in the comment, the clear_output can only be used in a Notebook. If you do not use Notebooks, then you should remove the import and the call to.

Step 2 Input from the user with validation

Human-computer interaction is one of the most difficult parts to implement.

Why?

10 - Interactive Board Game

One way to do that is by using helper functions as follows by using conditional statements.

```
def get_position(board, marker):
    return board.index(marker)


def valid_move(board, input_str):
    if input_str not in ['-2', '-1', '1', '2']:
        return False

    move = int(input_str)
    pos = get_position(board, 'O')

    if pos + move < 0:
        return False
    if pos + move >= len(board):
        return False

    return True


def user_input(board):
    while True:
        input_str = input('Choose move (-2, -1, 1,
2): ')
        if valid_move(board, input_str):
            return int(input_str)

        print('invalid move')
```

We use an infinite while loop to get input.

10 - Interactive Board Game

Step 3 Check if user lost

Next we need to check if the user lost.

This can be done as follows.

```
def game_done(board, move, marker):
    pos = get_position(board, marker)

    if board[pos + move] != ' ':
        return True
    else:
        return False
```

Step 4 Update the board

Then we need to update the board.

```
def update_board(board, move, marker):
    pos = get_position(board, marker)

    board[pos] = ' '
    board[pos + move] = marker

    return board
```

Step 5 Make a computer move

This requires randomness. Python has standard library for that. How can you make a valid random move.

1. Make a random move (not necessarily valid)
2. Check if valid, if not valid go to 1 else done.

This can be implemented as follows.

```
from random import randrange

def get_random_move():
    while True:
        random_move = randrange(-2, 3)
        if random_move in [-2, -1, 1, 2]:
            return random_move

def get_computer_move(board):
    pos = get_position(board, 'X')

    while True:
        move = get_random_move()

        if pos + move < 0:
            continue

        if pos + move >= len(board):
            continue

        return move
```

It is also using continue as a great way to keep the flow simple in the while loop.

Step 6 Check if computer won

Now wait a minute.

How is the different from check if player lost?

It is only about the marker.

Luckily, we implemented it in a way where we set the marker as an argument.

This will make sense when you see the final code.

10 - Interactive Board Game

Step 7 Update board for computer

Again, this is just as updating it for the player, because we made the marker an argument.

Putting it all together

Now this is the power of modular programming.

Creating the final program is simple.

```
board = ['O', ' ', ' ', ' ', ' ', 'X']
moves = 0

while True:
    display_board(board)

    move = user_input(board)

    if game_done(board, move, 'O'):
        print('You lost')
        break

    board = update_board(board, move, 'O')
    moves += 1

    computer_move = get_computer_move(board)

    if game_done(board, computer_move, 'X'):
        display_board(board)
        print('Computer move', computer_move)
        print('You made', moves, 'moves')
        break

    board = update_board(board, computer_move, 'X')
```

Now how about that.

11 - Tic Tac Toe Game

Why create an interactive game as a Python developer

Implementing an interactive game, like Tic Tac Toe in Python, as a programmer can be an exciting and rewarding experience with several benefits.

Here are some of the most important reasons why implementing an interactive game, like the Tic Tac Toe game in Python, is valuable for a programmer:

- **Creative Expression**. Developing an interactive game provides programmers with a unique opportunity to express their creativity and imagination. Games often involve designing characters, creating worlds, and developing game mechanics, all of which require a high degree of creativity.

- **Problem-Solving**. Developing a game involves solving a wide range of problems, from designing the game's architecture to implementing the game mechanics. This process can help programmers develop their problem-solving skills and learn how to approach complex challenges.

- **Learning**. Developing an interactive game can be a great way to learn new programming languages, libraries, and tools. Game development often requires working with a wide range of technologies, from game engines to graphics libraries, which can broaden a programmer's skill set.

- **Collaboration**. Game development often requires working in a team, which can help programmers develop collaboration and communication skills. Building a game requires designers, artists, and developers to work together to achieve a shared goal, which can help programmers understand the importance of teamwork and collaboration.

11 - Tic Tac Toe Game

- **Engagement**. Games are interactive by nature, which makes them an engaging and immersive experience. Implementing an interactive game can help programmers understand how to build user interfaces that are easy to use and engaging, which is valuable in many other applications beyond games.

- **Marketable Skills**. Game development is a highly valued skill in the tech industry, and building a game can help programmers develop a portfolio that demonstrates their expertise in programming, problem-solving, and collaboration.

Overall, implementing an interactive game is a valuable experience for a programmer that can help them develop a wide range of skills, from creativity and problem-solving to collaboration and communication. Plus, the satisfaction of building a game that others enjoy playing can be an incredibly rewarding experience.

Project Description

Tic Tac Toe is a two-player game where each player takes turns placing X's or O's on a 3×3 grid.

The objective is to get three of your symbols in a row, either horizontally, vertically, or diagonally, while preventing your opponent from doing the same.

The game ends in a draw if the grid is full and neither player has won.

It is a simple and popular game often used to teach basic game theory and programming concepts.

The goal of this project is to implement a game, where the player is O and the computer is X. In this simple version, you implement the player **O** always start.

11 - Tic Tac Toe Game

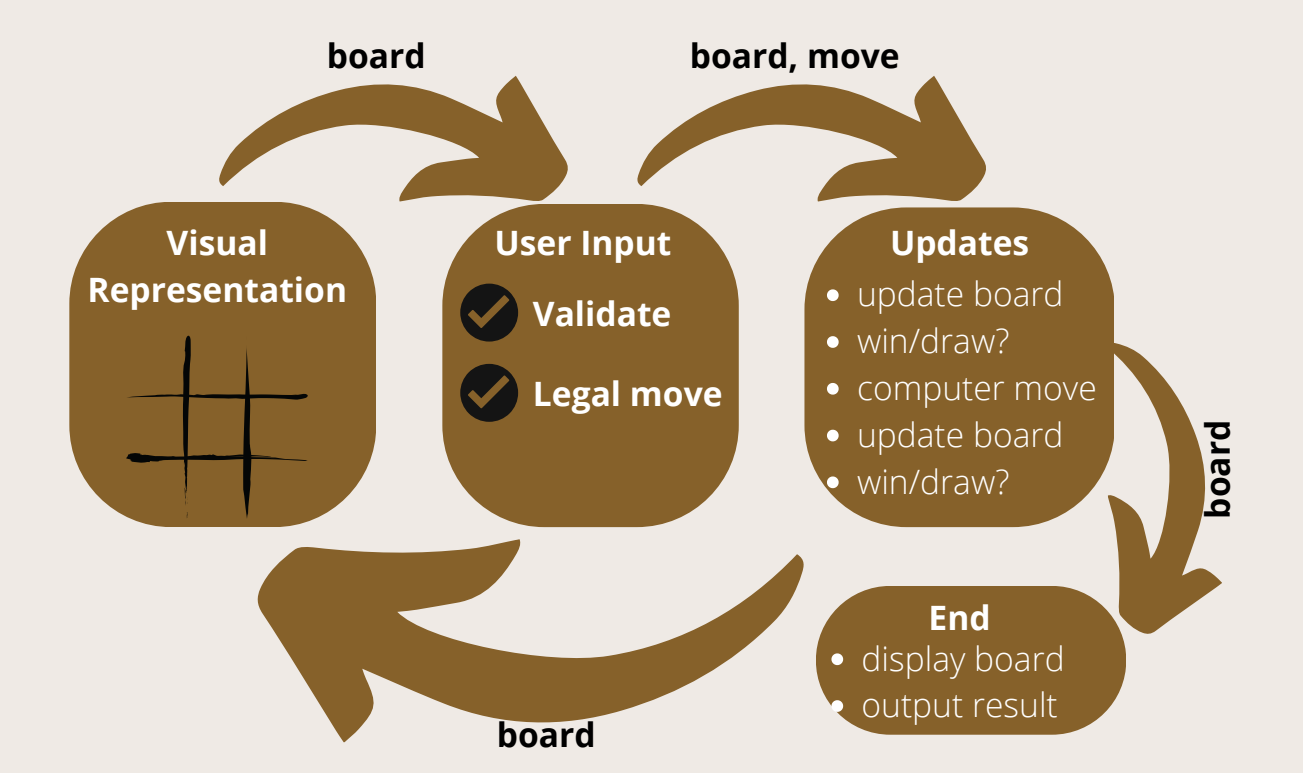
Design of Project

As we have learned, it is important to make a design of your project before you start to code. This is also true for implementing the Tic Tac Toe game in Python.

- It clarifies the project requirements, making it easier to understand the nature of what you need to implement.
- It gives structure to the project, making it easier to know what to code and how to connect the pieces.
- It saves time, as you have planned the full implementation, and is, therefore, less likely to get surprises.
- It facilitates collaboration, as you can branch out and implement pieces independently.

Not to forget, it makes your code easy to test in individual parts. One way to break down a game is by following the logical flow of the game.

It is important to understand that this flow could be considered as follows.



11 - Tic Tac Toe Game

This can be done as follows for a tic tac toe game.

0. Setup board
1. Display board
2. Input from the user (including validation)
3. Update board (place the marker for the player)
4. Check if the player wins or the game is done then output it and end.
5. Input from computer
6. Update board (place the marker for the computer)
7. Check if the computer wins or the game is done then output it and end.
8. Repeat from 1.

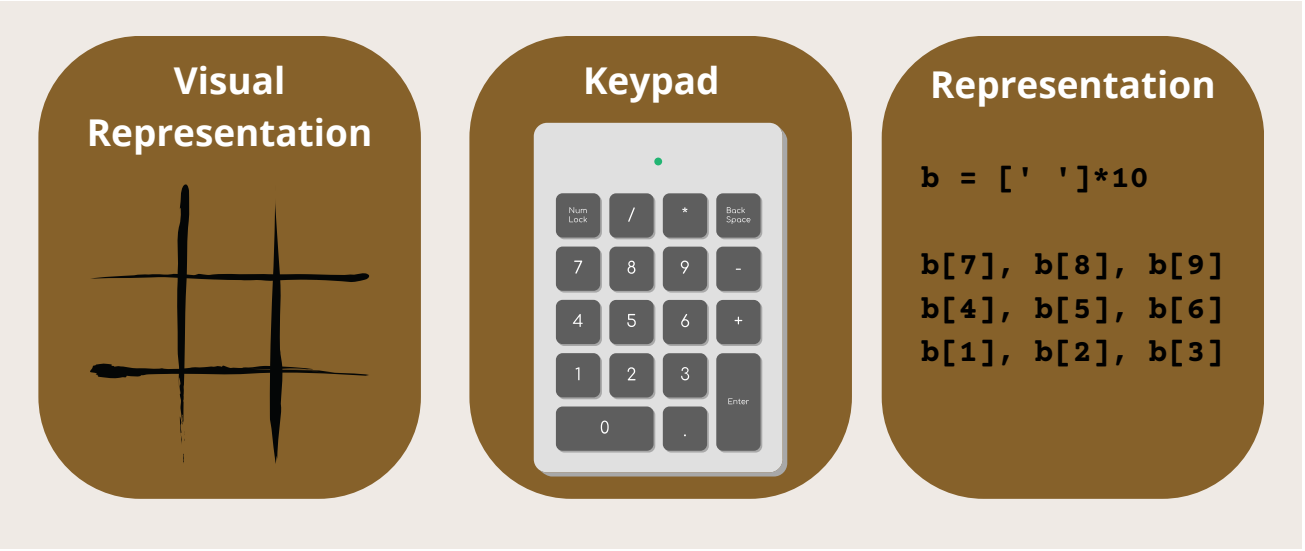
Now we are ready to implement the tic tac toe game in Python using functions.

Step 0 Setup board

While this can seem like a simple part of the project, it often has the biggest impact on the rest of the code.

How you represent the board will influence how other parts of the code will be implemented.

What we will use here is the simple keypad of numbers.



11 - Tic Tac Toe Game

You see, using the keypad as a way to represent the board is a great way to represent it. Later, you will also see how it can simplify other parts of the code.

Alternatively, you could have a list of lists. But try to think about how you would implement other parts of the code later. It would make it more complex.

Hence, we keep the board as follows.

```
board = ['#', '-', '-', '-', '-', '-', '-', '-', '-', '-']
```

Notice, we do not use the first entry (index 0) and keep a hash symbol there. This actually doesn't matter, because it will simplify the code as we proceed.

The only entries we use are 1 to 9.

Step 1 Display the board

Already when we need to display the board, we see the power of just having it represented as a keypad.

```
from IPython.display import clear_output

def display_board(board):
    clear_output()
    print(board[7], board[8], board[9])
    print(board[4], board[5], board[6])
    print(board[1], board[2], board[3])
```

This makes it clear and concise what the board looks like and is easy to understand as a programmer.

Notice this code is made for Notebooks, if you do not use Notebooks, you cannot use the important clear_output().

If you want to make a more fancy display, you are welcome.

11 - Tic Tac Toe Game

Step 2 Input from the user

User interaction is often a bit complex. The user may interact in the wrong way.

Therefore you often need validation of the input.

A great way to do that is by using while loops.

```
def valid_move(board, input_str):
    """
    Return true if move valid (1-9 and available).
    """
    if len(input_str) != 1:
        return False
    if input_str not in '123456789':
        return False
    move = int(input_str)

    if board[move] != '-':
        return False

    return True

def player_input(board):
    """
    Input player move. Prompts the user until valid move
    meaning (1-9) and position available.
    Returns the valid move.
    """
    while True:
        input_str = input('Input move (1-9): ')
        if valid_move(board, input_str):
            return int(input_str)

        print('Invalid move')
```

You see, the user is prompted until a valid move has been made. This is a great way to implement it, as it makes it easy for somebody reading the code as well.

Always think about the future person reading the code. How can you design and make the implementation easy?

11 - Tic Tac Toe Game

Step 3 Update the board

This is straightforward, but there is one thing to remember.

We have already validated that the player makes a valid move, therefore it is straightforward to implement this part.

```
def place_marker(board, marker, move):
    """
    Places the marker at position on board and returns it.
    """
    board[move] = marker
    return board
```

You may think creating a function for this is a bit too much, but it will simplify and make your final code more readable and easier to understand. The function name tells the reader of the code what the intention is, also, the doc string tells about that.

If you just put the code somewhere, the reader can see what happens, but not know what the intention was, making it difficult to understand.

Step 4 Check if the game is done

We need to check two things.

1. Are there any moves left (is the board filled)
2. Did the player win?

This can be implemented in two functions as follows.

11 - Tic Tac Toe Game

```
def is_done(board):
    """
    Returns True if the board is filled and game is done.
    """
    for pos in board[1:]:
        if pos == '-':
            return False

    return True

def player_won(board, marker):
    """
    Return True if player with marker has won.
    """
    # top row
    if board[7] == board[8] == board[9] == marker:
        return True
    # middle row
    if board[4] == board[5] == board[6] == marker:
        return True
    # low row
    if board[1] == board[2] == board[3] == marker:
        return True
    # left column
    if board[7] == board[4] == board[1] == marker:
        return True
    # middle column
    if board[8] == board[5] == board[2] == marker:
        return True
    # right column
    if board[9] == board[6] == board[3] == marker:
        return True
    # diagonal
    if board[7] == board[5] == board[3] == marker:
        return True
    # diagonal
    if board[9] == board[5] == board[1] == marker:
        return True

    return False
```

First notice, that is_done could be implemented more elegantly, but it does the job.

Secondly, player_won now uses the power of implementing the board as a keypad. It makes it easy to test.

11 - Tic Tac Toe Game

Step 5 Computer move

Now we need a move from the computer.

We will make a simple random move.

But how do we do that?

Simply, just make a random choice and see if it is valid.
You can get randomness by using Python's standard library.

```
from random import randrange

def get_random_postion():
    """
    Returns a random integer from 1-9 (both inclusive)
    """
    return randrange(1, 10)

def get_computer_move(board):
    """
    Returns a random move by the computer which is valid.
    """
    while True:
        random_move = get_random_postion()
        if board[random_move] == '-':
            return random_move
```

See, this is easy to implement and understand.

And actually, now we implemented all we need.

11 - Tic Tac Toe Game

Putting it all together

Now we can implement our tic tac toe game.

```
def play_game():
    """
    Plays a game of Tic Toc Toe with the computer.
    """
    # 0: setup board
    board = ['#', '-', '-', '-', '-', '-', '-', '-', '-', '-
']

    while True:
        display_board(board)
        pos = player_input(board)
        place_marker(board, 'O', pos)

        if is_done(board) or player_won(board, 'O'):
            display_board(board)
            print('Game done!')
            break

        c_pos = get_computer_move(board)
        place_marker(board, 'X', c_pos)

        if is_done(board) or player_won(board, 'X'):
            display_board(board)
            print('Game done!')
            break
```

You can make a more advanced version where it is random whether the user starts or not.

12 - Process IMDB Data Easily

Importance of Data Processing as a Python Developer

By learning to process IMDB data in Python you will learn the core skill of data processing.

Data processing is a critical aspect of modern computing and is essential for businesses, organizations, and individuals who need to work with large amounts of data.

Python is a popular language for data processing due to its ease of use, large and active community, and extensive libraries and frameworks.

Here are some of the key reasons why data processing is important and why Python is a great language for data processing:

- **Data processing helps organizations make better decisions**. In today's data-driven world, businesses and organizations rely on data to make informed decisions. Data processing allows them to analyze, manipulate, and transform large amounts of data to extract insights and make better decisions.

- **Python is a powerful language for data processing**. Python's simplicity, flexibility, and extensive libraries make it an ideal language for data processing tasks. Libraries like NumPy, Pandas, and Matplotlib provide powerful tools for data manipulation, analysis, and visualization.

- **Data processing can improve efficiency**. By automating data processing tasks, organizations can improve efficiency and reduce the time and effort required for manual data processing.

12 - Process IMDB Data Easily

- **Python is easy to learn and use**. Python is known for its simplicity and readability, making it easy for beginners to learn and use. This makes it an ideal language for data processing tasks, even for those without extensive programming experience.

- **Data processing can uncover insights and trends**. Data processing allows businesses and organizations to analyze large amounts of data to uncover insights, patterns, and trends that can inform future decisions.

Project Description

This project will teach you how to make data processing on IMDB data in Python.

We will consider a small subset of the IMDB movie database. The file consists of about 5,000 rows of data and is found here. In this simple project, we will filter the rows and only keep some of the metadata.

This is a classical task in the data processing.

The job is to process the 5,000 rows and create a new csv file with less metadata and only for movies rated above imdb_score 7.

Project Design

While this seems to be an easy task, it is always important to make a few thoughts before starting to code.

One goal is to break the code down into smaller steps that can be implemented isolated and have minimal functionality to it. This will keep the code easier to understand and maintain.

12 - Process IMDB Data Easily

One way to break it down is as follows.

1. Read all the data
2. Prepare all the data
3. Process all the data
4. Write the processed data

The idea is not to try to do it all at once, but to keep the code in these steps.

Step 1 Read all the data

The IMDB data is kept in CSV files and we will use Python to process it.

We work with CSV data and we want to read it into a list of dictionaries.

```
import csv

filename = 'files/movie_metadata.csv'

with open(filename) as f:
    csv_reader = csv.DictReader(f)
    records = list(csv_reader)
```

Now we have all the data in the list records.

Step 2 Prepare all the data

You will notice, that this will keep all values in the dictionaries as strings.

We need to convert the imdb_score to floats in order to make the comparison.

12 - Process IMDB Data Easily

This can be done as follows using the type conversion function.

```
for record in records:
 record['imdb_score'] = float(record['imdb_score'])
```

Now the entry is converted to float and we can compare it as floats.

Step 3 Process the data

As we only need to keep some records, a great way is to create a new lists to keep them.

This makes the code easier to understand and maintain.

```
processed_records = []

for record in records:
    if record['imdb_score'] > 7:
        new_record = {
            'movie_title': record['movie_title'],
            'imdb_score': record['imdb_score']
        }
        processed_records.append(new_record)
```

Now we have all the data we need to write.

12 - Process IMDB Data Easily

Step 4 Write the processed data

Again, remembering how to work with CSV files.
You need to tell the fieldnames and write the header.

But all the records can be written at once.

```
with open('best_movies.csv', 'w') as f:
    csv_writer = csv.DictWriter(f, fieldnames=
['movie_title', 'imdb_score'])
    csv_writer.writeheader()
    csv_writer.writerows(processed_records)
```

While this is a simple project, it demonstrates the importance of keeping the code in steps that makes everything easy to understand and maintain.

Better do less in each step than thinking about performance and speed.

13 - Text Processing

Implement text processing as a Python developer

As a Python developer, it is important to be proficient in text processing because text data is ubiquitous in the modern world.

Many applications, websites, and platforms rely on text data as a primary source of information.

Text processing involves manipulating and analyzing text data, which can range from simple operations like counting the frequency of words in a document to more complex tasks like natural language processing and sentiment analysis.

Here are a few reasons why text processing is important for Python developers:

- **Data analysis**. Text processing is an important part of data analysis. Python developers can use text processing techniques to extract valuable insights from large volumes of unstructured data.

- **Automation**. Python can be used to automate many repetitive tasks that involve text processing, such as cleaning and standardizing text data.

- **Natural Language Processing (NLP)**. NLP is a rapidly growing field that involves analyzing and understanding natural language data. Python is a popular language for NLP due to its extensive libraries and tools for text processing.

- **Machine learning**. Text data is often used as input for machine learning models. Python has a wide range of machine learning libraries that are well-suited for working with text data.

Overall, text processing is a critical skill for Python developers who want to work with data, automate tasks, or build applications that involve natural language processing or machine learning.

13 - Text Processing

Project Description

Consider the file files/bachelor.txt.

What are the most likely words after the name Holmes occurs in the text.

Examples

```
...
     friend Sherlock Holmes had a considerable share in clearing
...
     still sharing rooms with Holmes in Baker Street, that he
...
```

In the two examples the word after Holmes is had and in.

How to solve this

There are many ways to solve this problem, but first, let's understand some of the obvious challenges.

- The text is in lines, and Holmes might be the last word on a line.
- There might be symbols after Holmes, like commas, punctuation, or similar.
- Uppercase and lowercase words should probably count as the same word.

Step 1 Read and split the content into words

A great way to deal with the issue of having multiple lines is to divide them into words.

13 - Text Processing

Luckily, Python has made that easy for you to read files.

```
# Read all the content
filename = 'files/bachelor.txt'
with open(filename) as f:
    content = f.read()

# Split it into words
words = content.split()
```

Now you have a list of words.

Step 2 Simplifying and counting

The next step could be to investigate the words.

You will notice words in uppercase, words with quotes, or whatever you want to remove.

Counting occurrences can be done easily with dictionaries.

```
freq = {}

# Used to record if the previous word was Holmes
last_word_holmes = False

# Iterate over all words
for word in words:
    # If last word was Holmes
    if last_word_holmes:
        last_word_holmes = False

        # Remove special characters
        word = word.replace("'", '').replace('"', '')
        word = word.replace(',', '').replace('.', '')
        word = word.lower() # Change to lowercase

        # Update the number of occurences
        freq[word] = freq.get(word, 0) + 1

    if 'Holmes' in word:
        last_word_holmes = True
```

Step 3 Displaying the counts

Let's say we only care about words occurring more than one time. Then we can do that as follows.

```
for word, count in freq.items():
    if count > 1:
        print(word, count)
```

This example has showed you how to process words easily by using split() and use replace() to remove characters.

Finally, how to use a dictionary to keep count.

14 - Acronym Generator

Implement an Acronym Generator in Python

A Python programmer can learn several important skills and concepts from making an acronym generator:

- **String manipulation**. Working with strings is a core aspect of programming an acronym generator. A programmer will learn to extract and manipulate strings to create new words and acronyms.

- **Loops**. An acronym generator may require the use of loops to iterate through strings and characters, such as when identifying the first letter of each word to form an acronym.

- **User input**. The acronym generator is designed to take user input, a programmer will learn how to handle and process input from users, including error handling.

- **Functions**. An acronym generator can be broken down into smaller functions to make the code more modular and reusable. A programmer will learn how to create functions that perform specific tasks and can be used in other program parts.

- **Documentation**. Writing clear and concise documentation is essential for any program, including an acronym generator. A programmer will learn how to document their code effectively, making it easier for others to understand and use the program.

Project Description

An acronym is a word created by combining the first letter or syllable of each word in a phrase to create a new, single word.

Here are a few examples of popular acronyms:

- **FOMO** fear of missing out
- **GIF** graphics interchange format
- **PIN** personal identification number

14 - Acronym Generator

Project Design

While this is a small project. an acronym generator in Python, it is always a good idea to break it down.

One simple way to do that is to break it down into functions of the natural steps in the process.

1. Input from the user
2. Create the acronym
3. Display the result

This makes a great structure for your project.

Step 1 Input from the user

The purpose is to get a phrase from the user.
One way to do that is to use the input function.

```
def get_input():
    """
    Prompts the user for an input phrase
    and returns it as a string.
    """
    input_str = input('Input word phrase: ')
    return input_str
```

A great thing about breaking it down into functions, is, that you can test each function isolated.

Step 2 Create the acronym

This is the core of the project.
Creating the acronym.

This can be done with a loop over the words. The function split() returns all the words from a string in a list.

15 - Password Generator

Implement a password generator

Implementing a password generator in Python will teach you new skills.

- **String manipulation**. A password generator typically involves manipulating strings in order to create random sequences of characters that meet certain criteria (e.g., length, complexity). As such, a programmer will gain experience with string concatenation, slicing, and other operations that are commonly used in string manipulation.

- **Random number generation**. Generating random numbers is a core component of any password generator. A programmer will learn how to use Python's built-in random module to generate random integers and other values.

- **Conditional statements**. A password generator will typically need to make decisions based on the user's input (e.g., if the user specifies a length of 8 characters, the generator should create a password that is exactly 8 characters long). This will require the use of conditional statements (e.g., if, else) to handle different cases.

- **Loops**. A password generator may need to repeat certain operations (e.g., generating random characters) multiple times in order to create a password that meets the desired criteria. This will require the use of loops (e.g., while, for) to iterate over a sequence of characters or numbers.

- **Secure password generation**. Creating a secure password that cannot be easily guessed by an attacker is a key goal of any password generator. A programmer will learn about common password security best practices (e.g., avoiding common words, using a mix of uppercase and lowercase letters, including numbers and symbols) and how to implement them in code.

Are you ready?

15 - Password Generator

Project Description

Create a password generator in Python.

The password should in this format (with a dash - every 4th character).

- `vr)5-x?C(-8FNh-SkfD`

The password should have:

- at least one lowercase character: abcdefghijklmnopqrstuvwxyz
- at least one uppercase character: ABCDEFGHIJKLMNOPQRSTUVWXYZ
- at least one decimal: 01234567890
- at least one special character: !@#$%^&*()?

How do you do that?

Step 1 Getting the characters

A simple way to keep track of which characters are needed in the password is to simply have them in each variable.

This makes the code understandable later.

```
lower = 'abcdefghijklmnopqrstuvwxyz'
upper = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ'
decimal = '01234567890'
special = '!@#$%^&*()?'
```

Step 2 Checking if the password is strong

A strong password should have at least one of each of the 4 groups of characters: lower, upper, decimal, and special.

15 - Password Generator

To not complicate code, a great way is to keep the functionality simple by creating your own simple functions using loops and conditionals.

```
def contains(letters, password):
    for letter in letters:
        if letter in password:
            return True
    return False


def strong_password(password):
    if not contains(lower, password):
        return False
    if not contains(upper, password):
        return False
    if not contains(decimal, password):
        return False
    if not contains(special, password):
        return False

    return True
```

The logic is simple. For each group check if the password contains the type of characters. If not, return False. If it passes all tests, then the password must be strong.

Step 3 Generate strong passwords

Now to the most difficult part.

How do we generate a strong password?

The simple answer is, to generate random passwords until one is strong.

15 - Password Generator

To do that we need randomness.

```
import random

legal_chars = lower + upper + decimal + special

while True:
    password = ''
    for _ in range(16):
        password += random.choice(legal_chars)

    if strong_password(password):
        password = password[:4] + '-' +
password[4:8] + '-' + password[8:12] + '-' +
password[12:]

        print(password)
        break
```

You see, keep making them until one succeeds. This way you don't compromise the randomness in the password generator.

16 - Anagram Game

What programming an anagram game will teach you

On your journey to becoming a great Python programmer, you need to dive into projects that match your current level, like the anagram game.

An anagram game will teach you the following.

- **String manipulation**. An anagram game involves manipulating strings to create new words or phrases from the letters of a given word or phrase. As such, you will gain experience with string manipulation techniques such as slicing, concatenation, and sorting.

- **Input and output**. The game will require the ability to accept user input (i.e., the word or phrase to be scrambled) and output the resulting anagrams to the user. This will require the use of Python input() function and the print() function.

- **Looping** The game will need to iterate to prompt the user for guesses until correct.

- **Data structures**. An efficient anagram game will require storing the dictionary of words in a data structure such as a list or a dictionary for quick lookups. You will learn how to use data structures in Python to optimize the game's performance.

- **Randomness**. To make the game fun you need some randomness. This will teach you how to use randomness both to select a random word and to create a random anagram.

16 - Anagram Game

Project Description

What is an Anagram?

An anagram of a word is another word obtained by shuffling its letters.

Example

race and care are anagrams of each other.

What to do to implement an anagram game in Python
Create a program that will show the anagram of any word the user input. The user should guess it.

To simplify it a bit, we will make a scramble of the word. Hence, the anagram will most likely not be a correct word.

Step 0 Get some words

To keep it simple we will only use a short list of words.

This way we also have a chance to guess the anagrams.

```
words = [
    'atrocity',
    'fanatical',
    'pensive',
    'respite',
    'discordant',
    'eloquent',
    'encompass',
    'imperceptible',
    'insuperable',
    'stealthily',
    'impassive',
]
```

These are the words we will use to create anagrams.

16 - Anagram Game

Step 1 Get a random word

We can use random to get a random word from the list.

```
import random

word = random.choice(words)
```

Now word contains a random word from the list of words.

Step 2 Shuffle the characters

This will not generate a true anagram, but shuffle the characters of the word to use.

```
letters = list(word)
random.shuffle(letters)
anagram = ''.join(letters)
```

Now we have the anagram.

Step 3 The game

To create the game we need a loop to keep querying the player.

```
while True:
    print(anagram)

    guess = input()

    if guess == word:
        print('correct')
        break

    print('incorrect')
```

Now that is powerful.

17 - Valid Parentheses

Implement a valid parentheses checker in Python

Keep your skills sharp and start learning Python by creating a valid parentheses checker. There is no better way than to make projects.

To create a valid parentheses checker you will learn.

- **Syntax and semantics of Python language**. Creating a parentheses checker requires a good understanding of Python's syntax and semantics. You will learn how to use conditional statements, loops, and string manipulation functions to check if parentheses are balanced or not.

- **Problem-solving skills**. Building a parentheses checker requires problem-solving skills. You will need to think through the logic of how to check if parentheses are balanced or not and how to handle different scenarios such as nested parentheses.

- **Algorithmic thinking**. You will need to develop an algorithm to solve the problem of checking for balanced parentheses. This involves breaking down the problem into smaller steps and designing a sequence of instructions to solve the problem.

- **Data structures**. You will likely use data structures such as stacks or queues to implement the parentheses checker algorithm. By building the checker, you will gain a deeper understanding of these data structures and how to use them effectively in your code.

- **Code organization**. Writing a valid parentheses checker involves creating functions and modularizing your code. This will help you understand how to organize your code in a way that makes it easy to read, maintain, and reuse.

17 - Valid Parentheses

Project Description

A valid parentheses string satisfies the following:

- Every opening bracket must have a matching closing bracket of the same type.
- The brackets should be closed in the correct order.

Examples of valid
- ()
- (())
- ()()
- (()())

Examples of invalid
- (()
-)(
- ())(
- ((()))

Advanced

Can you solve it if the string contains simple, curly, and square braces: ()[]{}

Examples of valid
- ()[]
- ([])
- (){}
- ([]{})

Examples of invalid
- ([)]
- (}(]
- ()(}
- ({][})

The advanced version is a classical problem to solve in job interviewss

17 - Valid Parentheses

Simple valid parentheses checker

If you think about it, all you need is to keep track of the following.

- How many parentheses are being opened?
- How many are being closed?

While this is not the whole story you need to add a bit more.

- Have a variable of how many opening parentheses are not being closed yet.
- Then if you see one opening parentheses, then add one.
- If you see closing parentheses, subtract one, if the value becomes negative, then it is not a valid string of parentheses.
- After parsing the full string of parentheses, the value of the variable should be zero.

This can be implemented as follows using variables, conditional, and loops.

```
def matching_parentheses(s):
    no_open = 0

    for c in s:
        if c == ')':
            no_open -= 1
        elif c == '(':
            no_open += 1

        if no_open < 0:
            return False

    if no_open != 0:
        return False

    return True
```

17 - Valid Parentheses

Advanced valid parentheses checker

The complexity suddenly becomes bigger.

Because now you need to keep track of the order of opening and closing parentheses.

A closing parenthesis needs to match the opening.

To accomplish that you can use a stack. A stack can be implemented by using a Python list.

```
def matching_parentheses(s):
    stack = []

    for c in s:
        if c in '([{':
            stack.append(c)

        if c in ')]}':
            if len(stack) == 0:
                return False

            top = stack.pop()

            if top == '(' and c != ')':
                return False

            if top == '[' and c != ']':
                return False

            if top == '{' and c != '}':
                return False

    if len(stack) != 0:
        return False

    return True
```

18 - Four in a Row Game

Implement a four-in-a-row game in Python

On a high level, you will learn about the following creating a four-in-a-row game in Python.

- **Logic and control flow**. A four-in-a-row game requires a lot of logical operations, such as checking for a win condition and determining the best move to make. By implementing these operations using functions and conditional statements, a Python programmer can learn how to write code that responds to different inputs and produces different outputs.

- **Data structures**. A four-in-a-row game involves storing and manipulating large amounts of data, such as the state of the game board and the positions of the game pieces. By using data structures such as lists and dictionaries, you will learn how to manage complex data sets and perform operations on them efficiently.

- **User interaction**. A four-in-a-row game requires a user interface that allows the player to interact with the game and make moves. This can be extended to make a GUI, here we will keep it in text mode.

Are you ready?

Project Description

Four-in-a-row is a two-player strategy game where the objective is to connect four of your pieces vertically, horizontally, or diagonally on a grid of six rows and seven columns.

Players take turns dropping their colored discs into one of the columns, with the disc occupying the lowest unoccupied space in that column.

The first player to connect four of their discs in a row wins the game.

How to implement that?

18 - Four in a Row Game

Project Design

We know that breaking a project down into smaller steps is a great way to make it easier to implement.

A simple way is to consider how a game is structured. One way to break the four-in-a-row game in Python down could be as follows.

1. The board data structure
2. Display board
3. Get player 0 move
4. Check if the game is won or the board is full
5. Display board
6. Get player X move
7. Check if the game is won or the board is full

If you think about it, you only need to implement the first 4 steps, then you can implement the game.

Step 1 The board data structure

The most important decision is how you want to represent the board.

This has an impact on the rest of the code in your project.

The board has 6 rows and 7 columns.

```
empty = '.'
board = [[empty]*7 for _ in range(6)]
```

Here we define a specific character empty we will use. This makes some code easier to read later.

18 - Four in a Row Game

Step 2 Display board

Here we use the Notebook clear_output. If you are not writing code in a Notebook, you cannot use it the import and you will have to leave it out.

But here we use loops and functions to accomplish what we want.

Obviously, if you want to make it more advanced you can do that.

```
from IPython.display import clear_output

def display_board(board):
    clear_output()
    print(' 0123456')
    print('+' + '-'*7 + '+')
    for row in board:
        print('|', end='')
        for col in row:
            print(col, end='')
        print('|')
    print('+' + '-'*7 + '+')
```

Step 3 Get the player move

Now if you implement this one with a marker, it can be used for both players.

A simple way is to break this down into two functions.

One that prompts the user until a valid move is given.

18 - Four in a Row Game

Another one is to check if a move is valid.

```
def valid_move(board, move):
    if len(move) != 1:
        return False

    if move not in '0123456':
        return False

    if board[0][int(move)] == empty:
        return True

    return False

def get_player_move(board, marker):
    while True:
        print('Player:', marker)
        move = input('Input move:')
        if valid_move(board, move):
            return int(move)
        print('Invalid move')
```

Step 4 Check if won or done

This is by far the most complex to write.

Let's break it down into functions.

To check if the game is done (but not won) is to check if the board is full. This can be done simply like the following.

```
def is_full(board):
    for col in board[0]:
        if col == empty:
            return False

    return True
```

Now check if the player has 4 in a row. This needs to be done horizontally, vertically, and diagonally.

18 - Four in a Row Game

Again, break it down into more functions. One way could be as follows.

```
def get_transpose(board):
    trans = []

    for i in range(len(board[0])):
        col = []
        for row in board:
            col.append(row[i])

        trans.append(col)
    return trans

def four_in_row(board, marker):
    for row in board:
        row_str = ''.join(row)
        if marker*4 in row_str:
            return True
    return False

def game_won(board, marker):
    # Check rows
    if four_in_row(board, marker):
        return True

    # Check columns
    trans = get_transpose(board)
    if four_in_row(trans, marker):
        return True

    # Check diagonal
    diag = []
    for idx, row in enumerate(board):
        diag.append([empty]*idx + row + [empty]*(6 - idx))
    diag_t = get_transpose(diag)
    if four_in_row(diag_t, marker):
        return True

    diag = []
    for idx, row in enumerate(board):
        diag.append([empty]*(6 - idx) + row + [empty]*idx)
    diag_t = get_transpose(diag)
    if four_in_row(diag_t, marker):
        return True

    return False
```

18 - Four in a Row Game

Combining it into a game

Now we just need to combine it all into a game.

```
empty = '.'
board = [[empty]*7 for _ in range(6)]

while True:
    display_board(board)
    move = get_player_move(board, 'O')
    board = place_marker(board, move, 'O')
    if game_won(board, 'O') or is_full(board):
        display_board(board)
        print('Done')
        break

    display_board(board)
    move = get_player_move(board, 'X')
    board = place_marker(board, move, 'X')
    if game_won(board, 'X') or is_full(board):
        display_board(board)
        print('Done')
        break
```

Now isn't that great?

Python in 12 weeks

From Zero to Python Hero in Just 12 Weeks: Mastering Libraries and Frameworks for Web Scraping, PDF Generation, Excel Automation, and More!

- This Python program is **designed for beginners** who want to learn the language from scratch and quickly become proficient in **using libraries and frameworks**.
- In just **12 weeks**, you will learn the basics of Python programming, including syntax, data types, control structures, and functions.
- You will then move on to more advanced topics, such as **object-oriented programming**, **file handling**, **web scraping**, and **data analysis**.
- We will teach you how to use popular Python libraries and frameworks, including **BeautifulSoup**, automate tasks, generate **PDFs**, and manipulate **Excel** files.
- This program includes **hands-on projects** and **real-world examples** that will help you apply your newfound knowledge and build a portfolio of work to showcase your skills.

By the end of the program, you will have the **confidence and expertise to use Python libraries and frameworks** to solve complex problems and build powerful applications.

Get 67% discount and join

Make a change in your life

- **Learning Python in 12 weeks** requires a commitment to consistent practice and a willingness to challenge yourself.

- By following the above guide, you can **acquire a solid foundation in Python programming** and gain hands-on experience with coding exercises and projects.

- Learning **Python is a valuable investment** in your personal and professional development, offering opportunities to build web applications, analyze and visualize data, and build machine learning models, among other applications.

- With **dedication and perseverance**, you can become proficient in Python and leverage its vast library ecosystem and community resources to build sophisticated applications.

- **Start your journey today** and in just 12 weeks, you'll be surprised at how much you've learned and how far you've come in your Python programming skills.

Rune