

# Python in Plain Terms

## Contents

- Acknowledgments
- Python Introduction
- Google Colab
- Chp-1: Variables
- Chp-2: Input and Output
- Chp-3: Numbers
- Chp-4: Strings
- Chp-5: Conditionals
- Chp-6: Iterations
- Chp-7: Tuples
- Chp-8: Lists
- Chp-9: Functions
- Chp-10: Sets
- Chp-11 Dictionaries



Many people often ask me how to learn Python. They come from different backgrounds, ages, and education levels, with most having no coding experience. They're seeking a resource that's easy to understand without getting into technical jargon. That's the purpose of this online book. It explains

[Skip to main content](#)

Python concepts using simple language so that anyone, regardless of their field or expertise, can grasp it. After using this book, you'll be able to understand and write medium-sized Python code.

---

There is a separate section on how to use Google Colaboratory to write and execute Python code. However, the code in this book can be used in any major Python editor. It is easy to use Google Colab because you do not need to install any program on your computer. It is also possible to write your code on a tablet and store it in your Google Drive. Also, Google Colab makes it possible to share your code and work on code as a group.

---

Each chapter of this book covers a Python subject along with optional items. You can skip the parts that might be too detailed for you. However, it is essential to write and execute all codes in this book yourself. Whenever you see code in this book, try to write it in your code editor and execute it. If possible, make some small changes to check your understanding. You can copy/paste the code, but for beginners, it is better to write the code yourself.

---

Additionally, the book includes debugging, output, and coding questions tailored for different proficiency levels.

- Debugging: Identify errors within a given code.
  - These questions help you understand the syntax better and have an idea about the most common mistakes.
  - In these questions, first try to find mistakes, then think about how to solve them.
  - It is strongly recommended to write the correct code and execute it just to be sure it works well.
- Output: Determine the output produced by a given code.
  - By solving these kinds of questions, you will be able to understand each small detail of a code.
  - The best way to solve these questions is to work on paper and not look at the solution until being sure about your answer.
  - Also, it is a great practice to try to change this code and write it in a different, and if possible, better or shorter way.
- Coding: Write code for specific tasks.
  - This is the most important part to practice your coding skills with various types of questions.
  - The key point here is not looking at the solutions until having a solution, which might be even a

- Please try to solve the questions in these sections by using the information only in the preceding chapters.
  - Keep in mind that there might be more than one solution for such kind of questions. The solutions in this book might not be the best ones because simple solutions have been chosen for this textbook.
- 

This book will be updated very often, and a project chapter will be added soon.

This book is prepared by using Jupyter Book. Check out [the Jupyter Book documentation](#) for more information.

**Yusuf Danisman**, the author, is a faculty member in the Mathematics and Computer Science Department at Queensborough Community College, CUNY. He obtained his PhD in Mathematics from The Ohio State University and worked at the University of Oklahoma before joining CUNY. He teaches Mathematics, Statistics, Computer Science, and Data Science courses. Dr. Danisman is currently researching Financial Machine Learning and Geometric Deep Learning. He is also a steering committee member of the [Northeast Big Data Innovation Hub](#).

## Acknowledgments

For Meryem and Sumeyra,

I am deeply grateful to **Hamza Ali Danisman** for his careful review and helpful comments during the preparation of this textbook. I also want to thank the QCC STEM program and its director, **Anna Lee**, for her technical support during the preparation of this book.

## Python Introduction

- [Python](#) is one of the most commonly used programming languages.
- It was created by Dutch programmer [Guido van Rossum](#) in the late 1980s.
- He named it Python inspired by the BBC show [Monty Python's Flying Circus](#).

## Advantages

[Skip to main content](#)

- Example: Utilizing indentation (spaces) instead of parentheses results in clean code.
- The following two code checks whether the integer 15 is an even or odd number.
- Even if you do not understand what these two pieces of code are doing, you can see that Python has a simpler syntax.

Python	Java
<pre>number = 15  if number % 2 == 0:     print("The number is even.") else:     print("The number is odd.")</pre>	<pre>public class IfStatementCode {     public static void main(String[] args) {         int number = 15;          if (number % 2 == 0) {             System.out.println("The number is even.");         } else {             System.out.println("The number is odd.");         }     } }</pre>

- Example: No need to declare a variable and its type before using it.
  - In Java, you need to specify that age is an integer, name is a character (similar to a string in Python), game\_over is a boolean, and weight is a float. There is no such requirement in Python because Python can understand the type from the value.

Python	Java
age=25	int age =25;
name = 'mike'	char name ='mike';
game_over=True	boolean game_over=true;
weight=120.75	float weight=120.75f;

## 2. Free and open source: Python is freely usable and distributable, including commercial purposes.

- You can freely download and install it to your computer.
- Online editors are also available for use without any installation.
- Python packages are developed by major companies and shared for everyone's use.
  - Tensorflow was developed by Google.
  - Pytorch was developed by Facebook.

[Skip to main content](#)

3. General-purpose programming language: You can write code for different purposes.

- Write game code
  - Develop programs for stores
  - Perform visualizations
- 

4. Rich Libraries: Python has rich collection of libraries encompassing of tools for various fields.

- Numpy: for scientific computing
  - Statistics: offers statistics tools
  - Pandas: for data wrangling and analysis
  - Matplotlib: for visualization
  - Keras: for constructing neural network models
  - Django: for develop websites
  - Flask: for online applications
- 

5. Object oriented: Build on the concept of objects.

- Utilizes simple and reusable parts like blueprints
  - Short code may encompass many hidden functionalities
- 

6. Community Support: Python has a vast and active community that can provide assistance.

- You can find answers to your rquestions on platforms like [stackoverflow](#).
  - Extensive resources available on platforms like [linkedin](#)
- 

7. Portability: Python code can be run on other platforms with little to no modifications.

- Works on virtual platforms
  - Windows, Unix, Linux, macOS
- 

8. Python code can be combined with components written in other languages like C++, Java.

9. Python can be seen as a combination of general-purpose languages (such as C++ and Java) and domain-specific languages (like Matlab).

[Skip to main content](#)

# Disadvantages

1. Python code is visible to anyone using the application, allowing for code to be copied or modified.
2. The execution speed of Python is slower compared to languages like C++. This is mainly due to the use of an interpreter instead of a compiler.
  - The interpreter translates Python code into machine code, enabling the computer to understand and execute the code
  - While a compiler translates the entire source code in a single run, an interpreter processes the source code line by line.
  - To understand the difference between interpreters and compilers, you can watch the following video.

```
from IPython.lib.display import YouTubeVideo
YouTubeVideo('_C5AHaS1mOA', width=500, height=300)
```

Interpreters and Compilers (Bits and Bytes, Episode...)



# Installing Python

- It is easier to install Python with Anaconda
- [Anaconda](#) is a free and open-source distribution of the Python.
- It installs Python.
- It simplifies package management and deployment by providing a comprehensive collection of

[Skip to main content](#)

- It comes with over 250 packages automatically installed.
- It includes popular Python packages like NumPy, Pandas, Matplotlib, SciPy.

## Integrated development environment (IDE)

- IDE provides tools to write codes in a better, easier, and faster way.
- IDE includes:
  - Code editor
  - Debugger
  - Code highlighting
  - Auto completion
  - Project Management
- Examples :
  - [Spyder](#): It comes with Anaconda.
    - It is free and open source.
    - Designed by and for scientists, engineers and data analysts.
  - [Pycharm](#): It is developed by Jet Brains which is professional and very advanced.
    - Its community edition is free.
    - Professional edition is free for students and educators.

## Jupyter Notebook

- [Jupyter Notebook](#) allows writing or running code in a web browser.
- It does not require internet access.
- It is free and comes with Anaconda.

## Google Colab

- Short for [Google Colaboratory](#).
- An online tool for writing and running code.
- A cloud-based computational environment or notebook.

[Skip to main content](#)

- Files are automatically saved to Google Drive.
- As it is cloud-based, it requires an internet connection.
- Colab notebooks are Jupyter notebooks that are hosted by Colab.

## Applications

- The following examples demonstrate what can be accomplished with Python.
- The aim of this section is to provide you with an idea of Python's capabilities.
- You're not expected to comprehend the code at this stage.

### Import Stock Data

- You can import historical stock data from Yahoo Finance.
- The following represents data for Apple Stocks.

```
import yfinance
df = yfinance.Ticker('AAPL').history()
df.head().round(2)
```

Date	Open	High	Low	Close	Volume	Dividends	Stock Splits
2024-01-03 00:00:00-05:00	184.22	185.88	183.43	184.25	58414500	0.0	0.0
2024-01-04 00:00:00-05:00	182.15	183.09	180.88	181.91	71983600	0.0	0.0
2024-01-05 00:00:00-05:00	181.99	182.76	180.17	181.18	62303300	0.0	0.0
2024-01-08 00:00:00-05:00	182.09	185.60	181.50	185.56	59144500	0.0	0.0
2024-01-09 00:00:00-05:00	183.92	185.15	182.73	185.14	42841800	0.0	0.0

[Skip to main content](#)

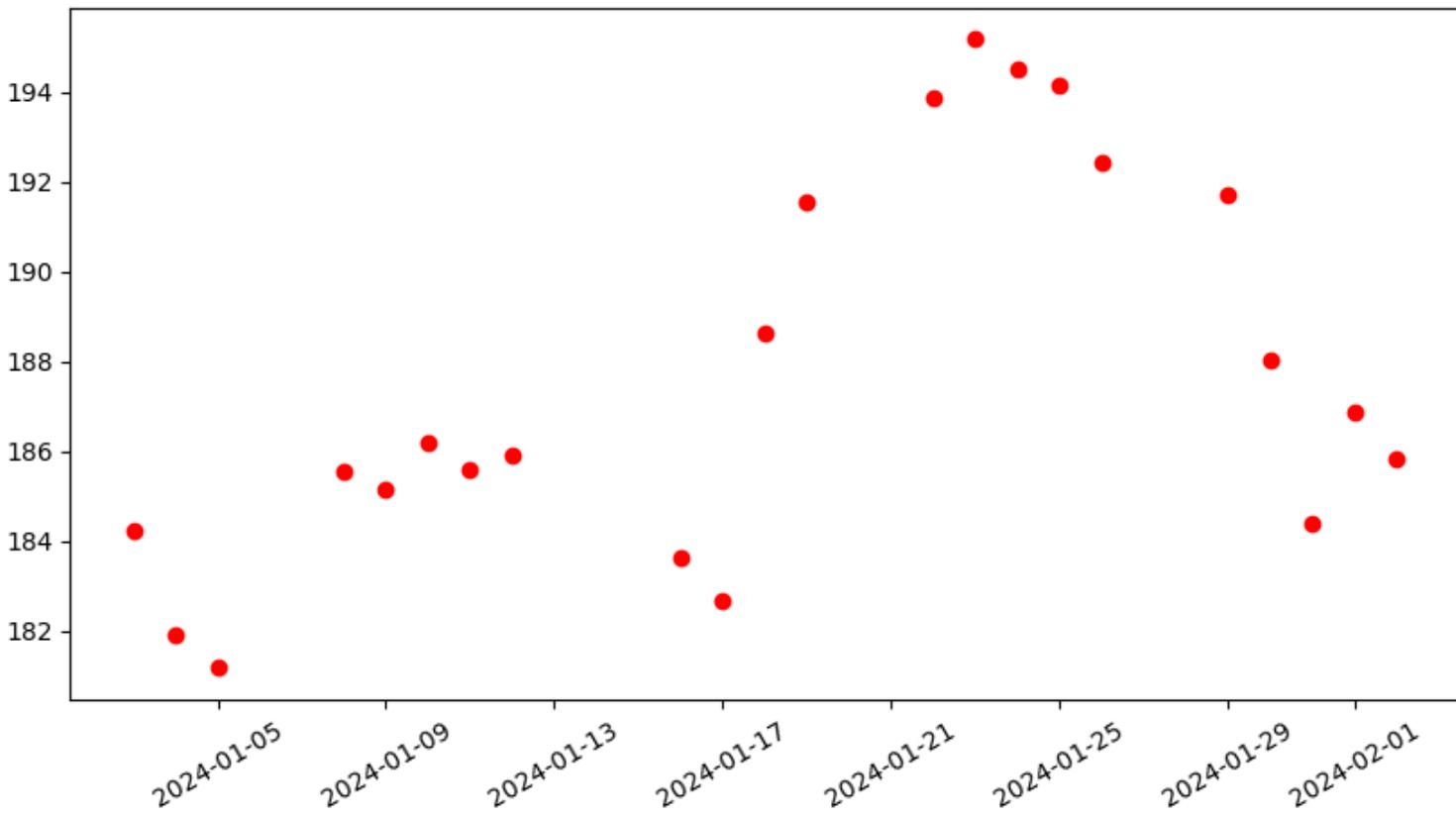
- You can import tables from Wikipedia websites.
- The following is the table of SP500 companies in the website  
[https://en.wikipedia.org/wiki/List\\_of\\_S%26P\\_500\\_companies](https://en.wikipedia.org/wiki/List_of_S%26P_500_companies)

```
import pandas as pd
pd.read_html('https://en.wikipedia.org/wiki/List_of_S%26P_500_companies')[0].head()
```

	Symbol	Security	GICS Sector	GICS Sub-Industry	Headquarters Location	Date added	CIK	Founded
0	MMM	3M	Industrials	Industrial Conglomerates	Saint Paul, Minnesota	1957-03-04	66740	190
1	AOS	A. O. Smith	Industrials	Building Products	Milwaukee, Wisconsin	2017-07-26	91142	191
2	ABT	Abbott	Health Care	Health Care Equipment	North Chicago, Illinois	1957-03-04	1800	188
3	ABBV	AbbVie	Health Care	Biotechnology	North Chicago, Illinois	2012-12-31	1551152	201 (188)
4	ACN	Accenture	Information Technology	IT Consulting & Other Services	Dublin, Ireland	2011-07-06	1467373	198

## Scatter Plot

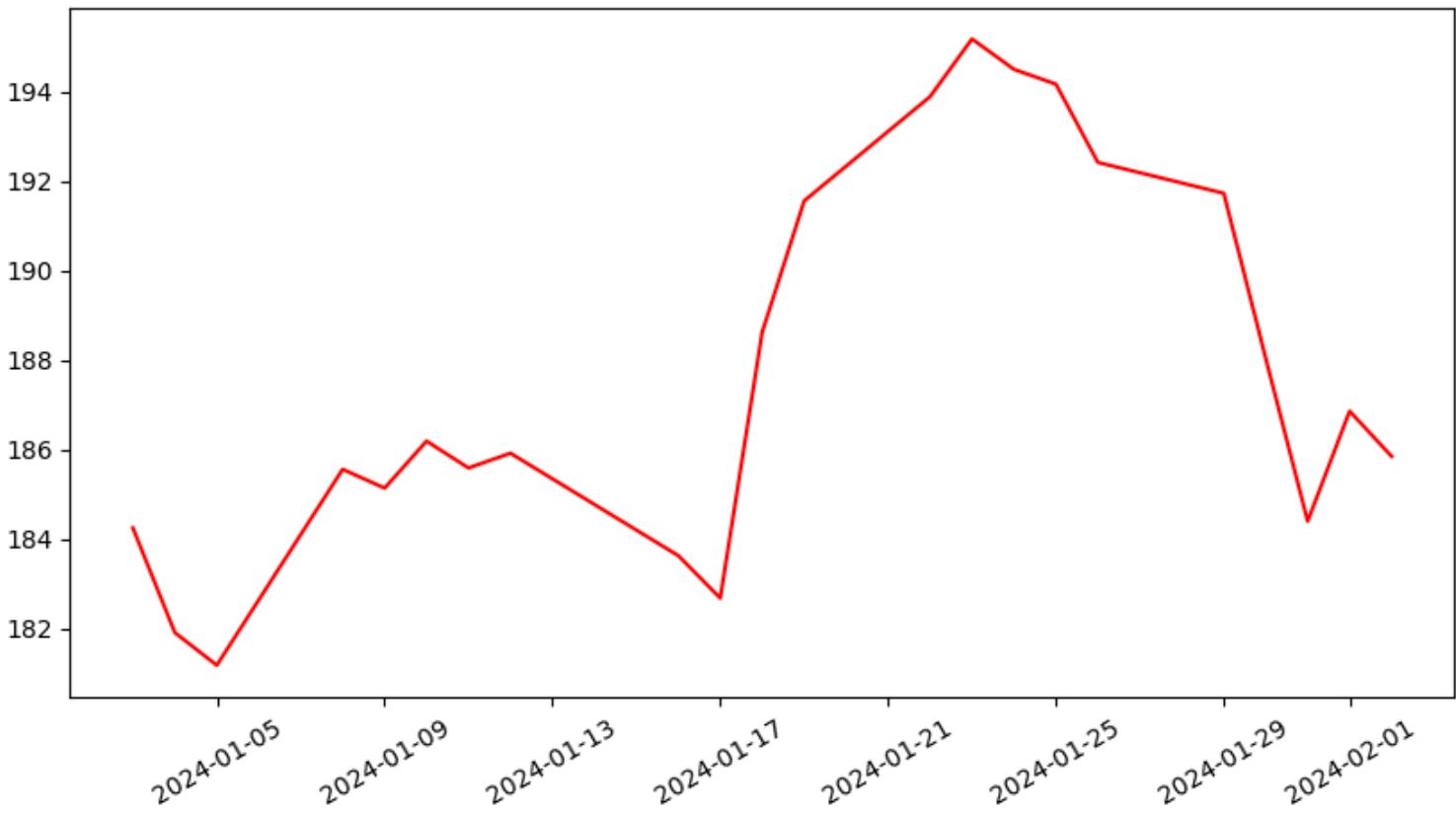
```
import matplotlib.pyplot as plt
plt.figure(figsize=(10,5))
plt.scatter(df.index, df.Close, color='r')
plt.xticks(rotation=30);
```



## Line Plot

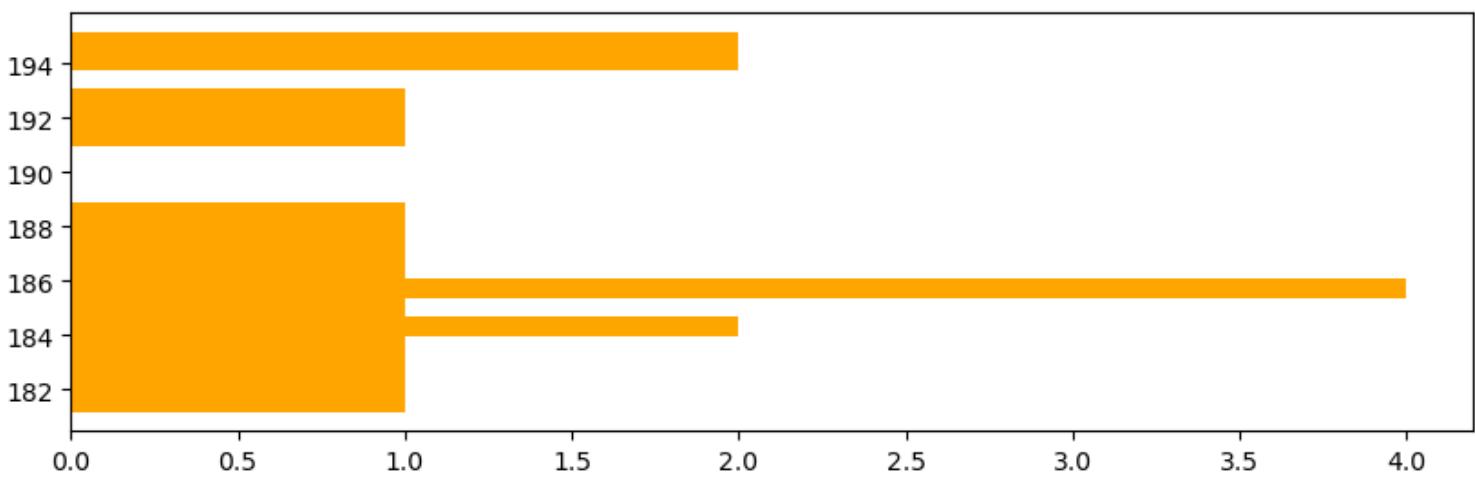
```
plt.figure(figsize=(10,5))
plt.plot(df.index, df.Close, color='r')
plt.xticks(rotation=30);
```

[Skip to main content](#)



## Histogram

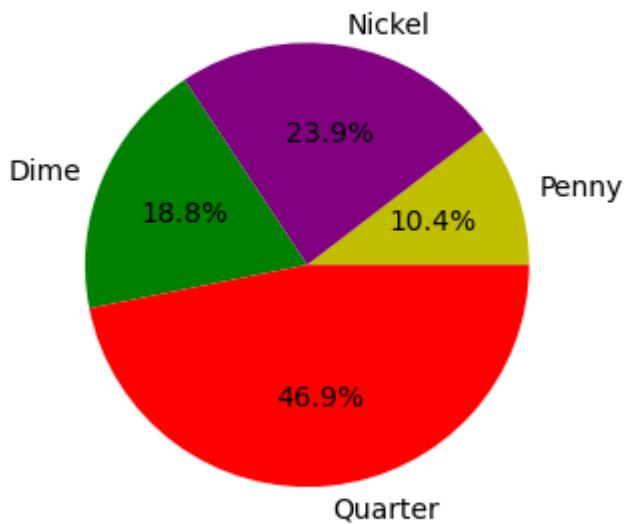
```
plt.figure(figsize=(10,3))
plt.hist(df.Close, bins=20, color='orange', orientation='horizontal');
```



## Pie Chart

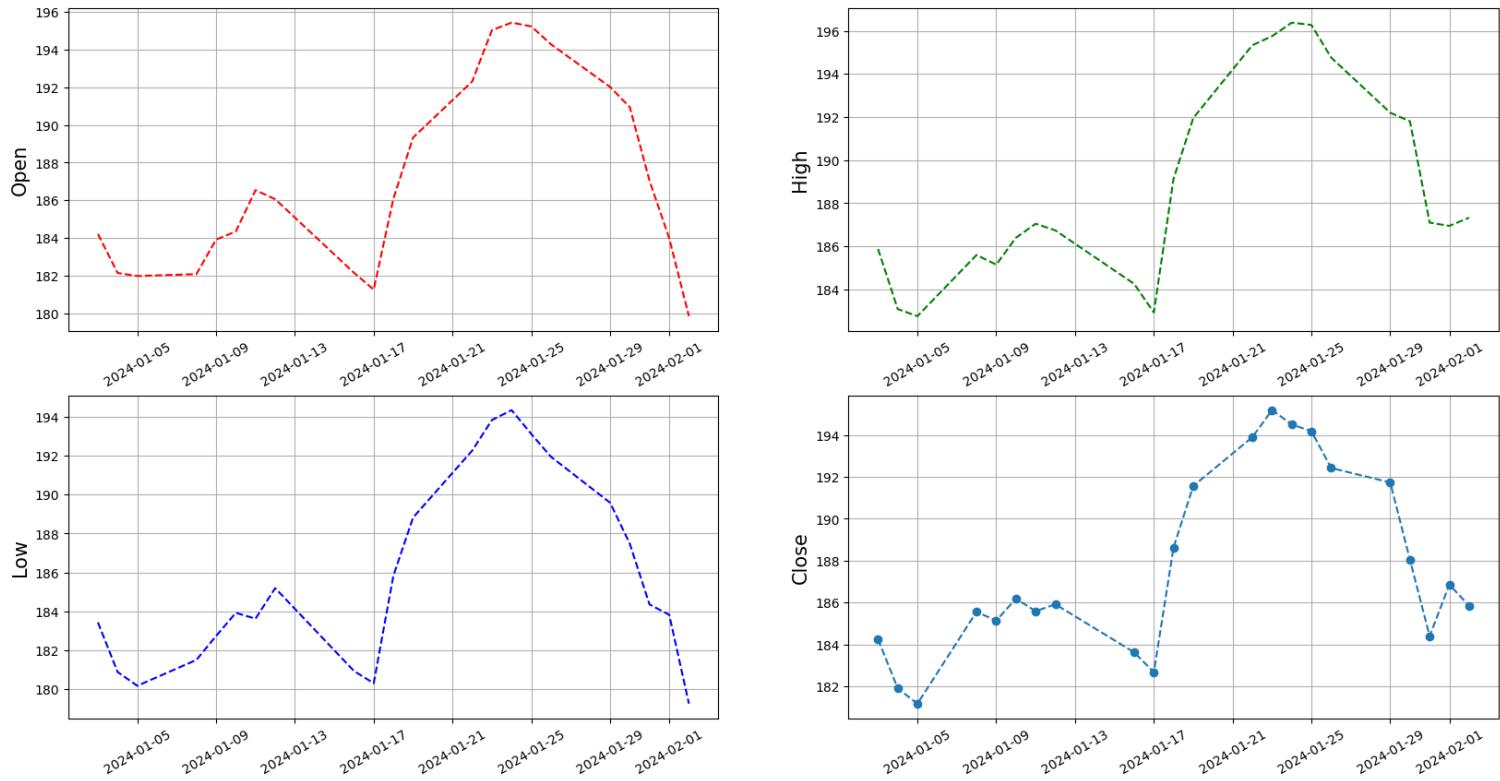
[Skip to main content](#)

```
number = [53, 122, 96, 239]
color_list = ['y', 'purple', 'g', 'r']
coins = ['Penny', 'Nickel', 'Dime', 'Quarter']
plt.pie(number, colors = color_list, autopct='%1.1f%%', labels = coins, radius=0.75);
```



## Multiple Plots

```
plt.figure(figsize = (20, 10))
color_set = ['r--', 'g--', 'b--', 'o--']
for i in range(1,5):
    plt.subplot(2, 2, i)
    plt.plot(df.iloc[:,i-1], color_set[i-1]);
    plt.ylabel(df.columns[i-1], fontsize=15)
    plt.xticks(rotation=30)
    plt.grid()
```



## Google Colab

Google Colab (Colaboratory) is a cloud-based notebook for writing and executing Python code.

- You need to sign into your Google account to execute code and save your notebook.
- There's no requirement to install any software on your computer.
- Sharing your notebooks is easy, and Colab files are automatically saved to Google Drive.

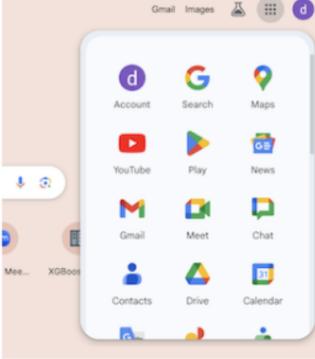
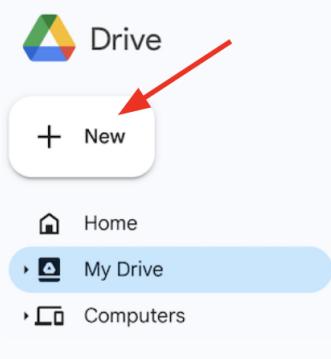
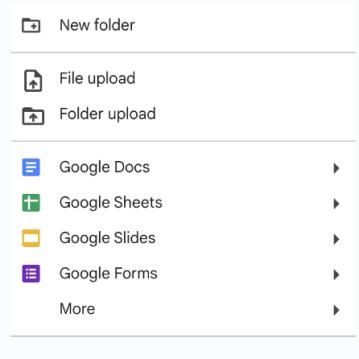
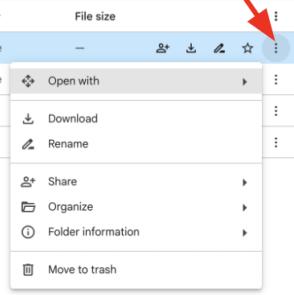
## Google Drive

Google Drive is used for storing files in the cloud.

- You can manage, modify, and share your files.
- By default, you have 15 GB of free storage, but there are also options for paid storage, such as 100 GB, 200 GB, and so on.
- Within your Google or Gmail account, you can access Google Drive and other applications by clicking on the Google Apps Icon, which has nine dots and resembles a square.
- By clicking on **+New** in the inner left corner of your Google Drive (located underneath the Drive

[Skip to main content](#)

- Upload a folder or file to Drive
- Create Google Docs, Google Sheets, Google Slides, Google Forms, etc.
- When you click on the **More Actions** icon (three vertical dots on the right), you will see options to modify your folder.
- You can move, download, share, rename, or delete your files and folders.

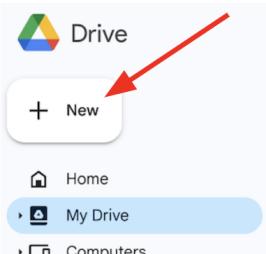
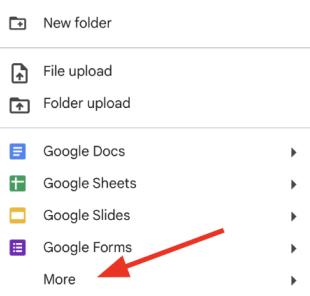
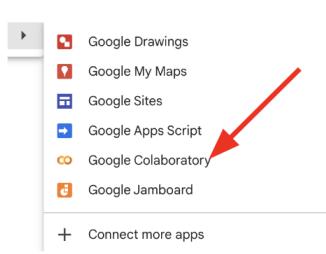
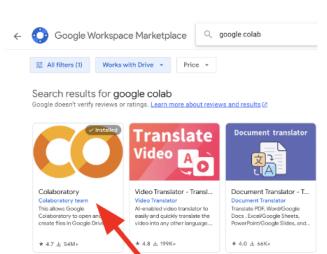
Google Apps Icon	+New	+New Menu	More Actions
			

## Installing Colab to your Google Drive

You can easily create Colab notebooks on your Google Drive by installing Colab onto your Google Drive (not onto your computer).

- Follow the following steps for installation:
  1. Click on **+New** in the upper left corner of your Google Drive (located underneath the Drive icon).
  2. Choose the **More** option at the end.
  3. In the new menu, if you see the Google Colaboratory icon, it means you already have Colab installed on your drive.
  4. If there is no Google Colaboratory icon, then click on **Connect More Apps.**
  5. Search for Google Colaboratory.
  6. Click on Google Colaboratory in the search results and install Colab on your drive.

[Skip to main content](#)

+New	+New Menu	More Menu	Search Results
			

## Create a new Colab notebook

Follow these steps:

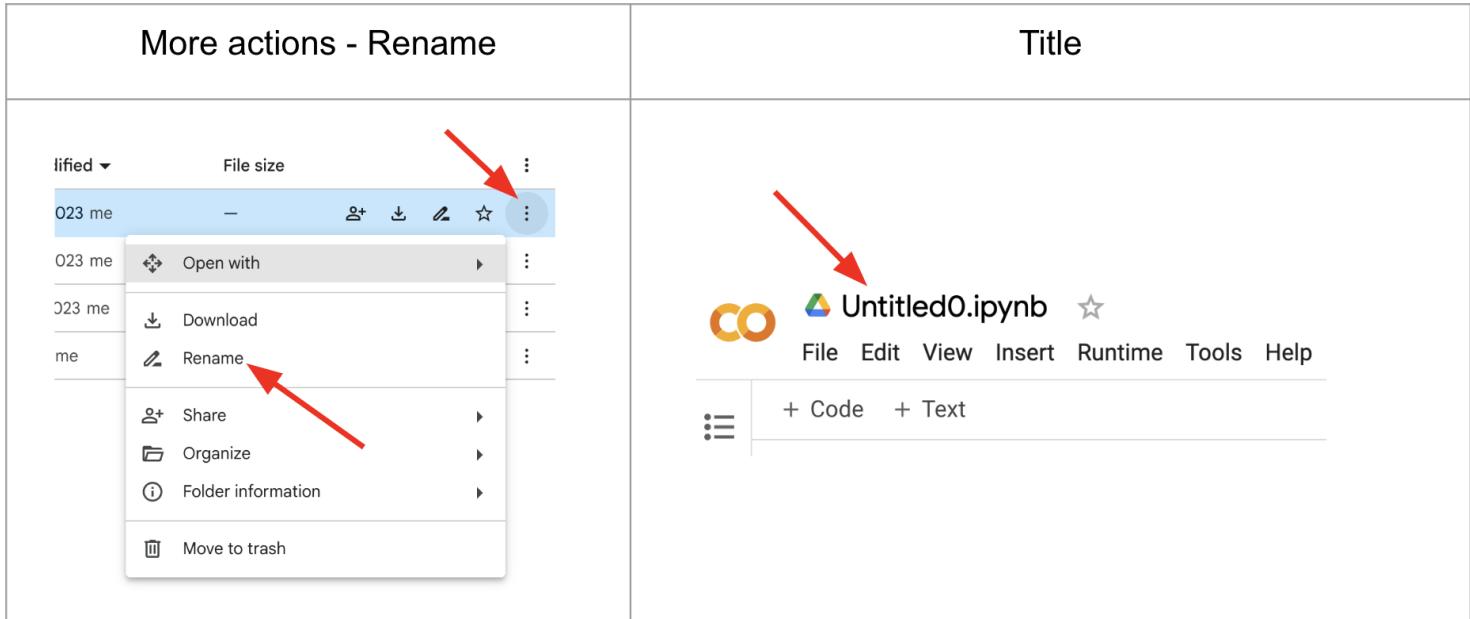
1. Click on **+New** in the upper left corner of your Google Drive (located underneath the Drive icon).
2. Choose the **More** option at the end.
3. In the new menu, click on **Google Colaboratory**.

Additionally, instead of using **+New**, you can right-click to access its contents.

## Rename a Colab File

- There are two ways to rename a Colab notebook:
  - Use the **More Actions** icon (three vertical dots on the right).
  - Open the Colab notebook (the default name is Untitled0) and click on its name in the upper left corner, next to the Google Drive icon.

[Skip to main content](#)



## Share a Colab File

To access the share options, you can do one of the following:

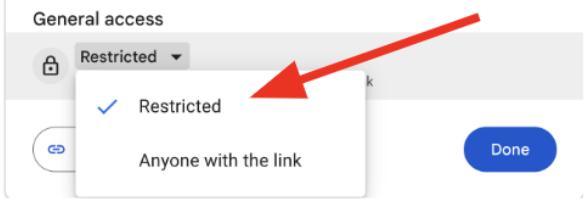
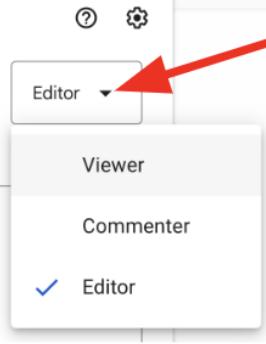
1. Use the **More Actions** icon (three vertical dots on the right).
2. Open the Colab notebook and click on **Share** in the upper right corner.

There are two ways to share a Colab notebook:

1. Enter the Gmail address of the person you want to share the notebook with.
2. Copy the link and share it, but change the last part from **Restricted** to **Anyone with the link**.

You have the following options to assign the person:

- Editor: Can change everything in the document.
- Commenter: Can write comments.
- Viewer: Can download a copy of the file but cannot modify your original file

Share Button	
Add Window for gmail entry	
Restriction	
Access Type Menu	

## More Tools

### Theme

There are three theme options:

- light (white background)

[Skip to main content](#)

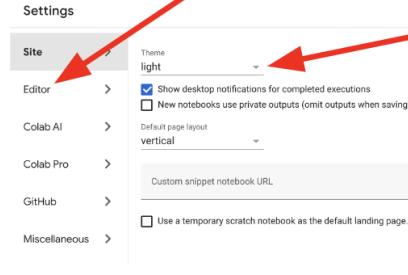
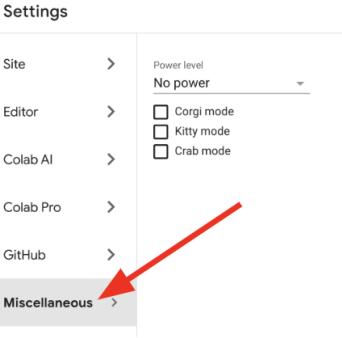
- adaptive (color depends on time)

Follow these steps on the toolbar to change the theme: **Tools** → **Settings** → **Theme**

## Power Level, and Kitty, Corgi, Crab

These are some fun tools to make the notebook more fancy:

- Go to **Tools** → **Settings** → **Miscellaneous**
- You can adjust the power level to add sparks when you write or execute code.
- You can add walking Kitty, Corgi, and Crab characters to your notebook.

Tools	Theme	Power Level, and Kitty, Corgi, Crab
<p>• Tools  Help All changes saved</p> <p>Command palette ⌘/Ctrl+Shift+P</p> <p>Settings </p> <p>Keyboard shortcuts ⌘/Ctrl+M H</p> <p>Diff notebooks</p>	 <p>Site Editor &gt; Theme light </p> <p>Editor &gt; Show desktop notifications for completed executions <input checked="" type="checkbox"/></p> <p>Colab AI &gt; Default page layout vertical</p> <p>Colab Pro &gt;</p> <p>Github &gt; Custom snippet notebook URL</p> <p>Miscellaneous &gt; Use a temporary scratch notebook as the default landing page <input type="checkbox"/></p>	 <p>Settings</p> <p>Site &gt; Power level No power</p> <p>Editor &gt; Corgi mode <input type="checkbox"/></p> <p>Colab AI &gt; Kitty mode <input type="checkbox"/></p> <p>Colab Pro &gt; Crab mode <input type="checkbox"/></p> <p>Github &gt;</p> <p>Miscellaneous &gt; </p>

## Cells in Colab

These are rectangular-shaped boxes that come in two types:

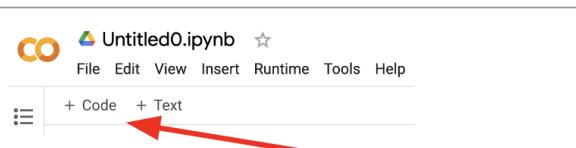
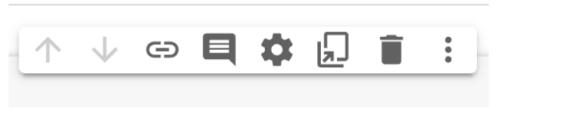
- Text Cell: Used for writing explanatory text.
  - Code Cell: Used for writing and executing Python code.
- 
- By default, a new notebook contains one code cell.
  - You can add, delete, and move cells up or down.
  - A Colab notebook is essentially a list of cells.

You can add a text or code cell by using the **+Code** and **+Text** buttons on the toolbar (upper left corner).

- You can use these buttons to add a cell below.

When you move the cursor to the upper (lower) middle of a cell, you will also see **+Code** and **+Text** buttons.

- You can use them to add a cell above (below).

+Code and +Text Buttons in toolbar	
+Code and +Text Buttons in a code cell	
Menu on the upper right corner of a code cell.	

## Delete, Move and Comment

When you click on a cell, a menu will appear in the upper right corner of the cell. Using this menu:

- You can delete the cell.
- You can move the cell up or down.
- You can add a comment to the cell.

## Short cuts

- **A**: Add a code cell above
- **B**: Add a code cell below
- **shift** + **Enter**: Execute a code cell

## Text Cell

[Skip to main content](#)

You can write explanatory text.

- By using tools covered in this section, you can compose well-organized texts.
- To edit a text cell, double-click on it.
- Click outside the text cell to exit the edit mode.
- Text cells in Colab use Markdown language syntax.
- When you start writing in a text cell, you will see another box on the right that shows the output.
- Text cells do not have an arrow on the left for execution.
- You do not need to memorize HTML code in this section; you can copy/paste it if needed.

## Heading

- Start the line with a hashtag (#) followed by a space.
- To make the header smaller, add more hashtags (#).

```
# Level-1 heading  
## Level-2 heading  
### Level-3 heading  
#### Level-4 heading  
##### Level-5 heading  
##### Level-6 heading
```

- The table of contents is located on the left side of Colab, marked by an icon displaying three rows of a dot and line.
- All headings appear in the table of contents according to their level.
- You can also use the  icon on the text cell toolbar to create a text heading.

## Center

- You can use the following HTML code to center text: `<center> text </center>`.
- Output:

text

- Use `**` (two asterisks) or `__` (two underscores) on both sides of the text to apply bold formatting.
- Alternatively, you can utilize the `B` icon on the text cell toolbar.
- The output of `How **are** you?` is
  - How **are** you?

## Italic

- Use `*` (one asterisk) or `_` (one underscore) on both sides of the text to apply italic formatting.
- Alternatively, you can utilize the `I` icon on the text cell toolbar.
- The output of `How *are* you?` is
  - How *are* you?

## Line Breaks

- Add 2 blank spaces or `\n` at the end of the line to start a new line.

## Color

- You can use the following HTML code to change the color of text: `<font color=red> text </font>`.
- Output: **text**

## Size

- You can use the following HTML code to change the size of text:  
`<font size=25> text </font>`.
- Output:

**text**

- You can use the following HTML code to change the color and size of text:

```
<font size=25, color=red> text </font>.
```

- Output:

text

## List

- Use a dash (-), asterisk (\*), or plus sign (+) followed by a space for an unordered (bullet) list.
- Use numbers followed by a period (.) and a space for a numbered (ordered) list.
- Indent to create a sublist.

The output of

```
- Africa  
- Europe
```

is

- Africa
- Europe

The output of

```
- Africa  
  1. Nigeria  
  2. Kenya  
- Europe  
  1. Germany  
  2. Italy  
  3. Spain
```

is

- Africa

[Skip to main content](#)

2. Kenya

- Europe

1. Germany

2. Italy

3. Spain

The output of

- Africa
  - 1. Nigeria
  - 2. Kenya
- Europe
  - 1. Germany
    - Munich
    - Dordmund
    - Bonn
  - 2. Italy
  - 3. Spain

is

- Africa
    - 1. Nigeria
    - 2. Kenya
  - Europe
    - 1. Germany
      - Munich
      - Dordmund
      - Bonn
    - 2. Italy
    - 3. Spain
- Instead of dash (-) you can also use asterisk (\*).

The output of

\* Africa

[Skip to main content](#)

is

- Africa
  - Europe
- 

## Highlight

- Enclose the text with one backtick on each side (`text`) to format it.
- In a light Google Colab theme, the highlight might not be very visible.
- For example, adding one backtick to each side of How are you? will display it as `How are you?`.

## Horizontal Line

- Use three asterisks (\*\*\* ) or underscores (\_\_\_\_) on a new line.
- In light mode, the horizontal line created by three asterisks (\*\*\* ) or underscores (\_\_\_\_) might not be very noticeable.

## Blockquote

- To create a blockquote, start a line with a greater than symbol (>).
- Use additional greater than symbols for nested blockquotes.

The output of

```
> aaa
```

is

```
aaa
```

The output of

[Skip to main content](#)

```
> aaa |\  
> bbb
```

is

```
aaa  
bbb
```

The output of

```
> aaa  
>> bbb
```

is

```
aaa  
  
bbb
```

The output of

```
> aaa  
> bbb  
>> ccc  
>>>ddd
```

is

```
aaa  
bbb  
  
ccc
```

[Skip to main content](#)

ddd

## Link

- Use the following Markdown syntax to create a link to a website: `[title](URL)`
- The output of `[Python](https://www.python.org)` is [Python](https://www.python.org)

## Table

- Use | (pipe) and - (dash) to create a table structure.

The output of

Gender	Age	State
Male	25	NY
Male	35	FL
Female	20	CA

is

Gender	Age	State
Male	25	NY
Male	35	FL
Female	20	CA

## Python code

- Use three backticks followed by the Python keyword to write Python code within a text cell,  
minimizing the appearance of a code cell

[Skip to main content](#)

The output of

```
``` python
for i in range(6):
    if i>2:
        print(i)
    else:
        print(True)
```

```

is

```
for i in range(6):
    if i>2:
        print(i)
    else:
        print(True)
```

## Mathematical Equations

- A text cell can execute Latex code.
- If you're not familiar with Latex, feel free to skip this part.
- The output of  $\int_1^5 x^5 dx$  is  $\int_1^5 x^5 dx$

## Add an Image

1. To import an image from your computer to a Colab notebook:

- Double-click on the text cell to edit it.
- Click on the image icon in the text cell's toolbar.
- Select the image you want to upload.
- You might see a lengthy and odd-looking text, but you can ignore it.
- Click outside the text cell to exit the edit mode.

2. To import an image from an URL use the following markdown syntax: `![] (URL)`

- The output of

[Skip to main content](#)



3. If you remove the `!` from the code above, it will create a link to the website of the image instead of displaying the image directly in the notebook.

- The output of `[python`

```
logo](https://www.python.org/static/community_logos/python-powered-h-100x130.png) is  
python logo
```

## Text Cell Questions

Please type the passages into text cells following the provided format.

### Question

#### My Life

I am 25 years old. I am from Spain. I have been to the following cities in Italy, during the following years and months

---

Europe

- Italy
  - Milan
    - 1993
      - March
      - April
    - 1994
      - January
      - June
      - July
  - Rome
    - 1. 1990
      - September
      - October
    - 2. 1990
      - August

---

I worked as an **engineer** in large companies including **Siemens**.

## Solution

### # My Life

I am 25 years old. \*I am from Spain. I have been to the following cities in Italy, during the following years and months\*

\*\*\*

Europe

- Italy

- Milan
  - 1993
    - March
    - April
  - 1994
    - January
    - June
    - July
- Rome
  - 1. 1990
    - September
    - October
  - 4. 1990
    - August

\*\*\*

I worked as an \*\*engineer\*\* in large companies including `Siemens`.



## Question

Education

# I have a BS degree in computer science.

I want to have a master degree in data science soon.

This table shows my graduation years and locations.

| School     | Graduation Date | Country |
|------------|-----------------|---------|
| Elementary | 1970            | Spain   |
| Middle     | 1973            | Spain   |
| High       | 1977            | Italy   |
| College    | 1981            | Germany |

## Solution

```
<center> Education </center>
```

```
<font color=red size=30> I have a BS degree in computer science. </font>
```

```
> I want to have a master degree in data science soon.
```

This table shows my graduation years and locations.

School	Graduation Date	Country
Elementary	1970	Spain
Middle	1973	Spain
High	1977	Italy
College	1981	Germany

[Skip to main content](#)

# Code Cell

You can write and execute Python code in code cells.

- An arrow on the left of a code cell allows you to run the code.
- Additionally, you can use the shortcut **Shift+Enter** to execute the code.
- A number in a square bracket on the left of executed code cells represents the execution order of that code cell.
- Warning: The order of a code cell in the notebook and its execution order can differ if you run another code cell above it

## print() function

- Displays output on the screen.
- In Colab notebooks, only the variable value in the last line is automatically displayed.
- To display any value in another line, you need to use the print() function.

```
# displays 5
print(5)
```

5

```
# notebook only displays the value in the last line of the code cell
5-1 # not displayed
3+7 # displayed
```

10

```
# print() displays the outputs
print(5-1) # displayed
print(3+7) # displayed
```

4  
10

[Skip to main content](#)

# Numbers

- You can use integers (...,-3,-2,-1,0,1,2,3,...) and decimal numbers called floats in a code cell.

## Algebraic Operations

- You can perform algebraic operations in code cells.
- The multiplication operator is `*` (asterisk).
- The division operator is `/` (forward slash).
- The exponent operator is `**` (double asterisk).

```
print(5+9)
```

14

```
print(10.4-4.5)
```

5.9

```
print(5*6)
```

30

```
print(10/2)
```

5.0

```
print(2**3)
```

8

[Skip to main content](#)

# Strings

- Strings are sequences of characters.
- They can be written inside single quotes, double quotes, triple single quotes, or triple double quotes.

```
print('Hello')
```

Hello

```
print("Hello")
```

Hello

```
print('''Hello''')
```

Hello

```
print(""""Hello""")
```

Hello

```
'3' # string '3' is not a number, it is only a character
```

'3'

```
# ERROR: you are trying to add a string (character) '3' and a number (5)
'3'+5
```

[Skip to main content](#)

- `String + String`: combines two strings
- `String * Integer` or `Integer * String`: makes copies of the string `Integer` many times.

```
# concatenation: combining two strings
print('Hello' + 'World')
```

HelloWorld

```
# repetition: # 10 Ms
print('M'*10)
```

MMMMMM

## Booleans

- There are only two boolean values: `True` and `False`.
- They indicate whether a condition is valid or not.
- Booleans will be covered in Conditionals chapter with details.

## Comments

- Use `#` to add notes to your program, explaining the purpose of the code.
- Comments are not executed but serve as explanations.
- Additionally, you can place comments to the right of the code.
- Comments make the code easier to read and understand.
- Comments help team members understand each other's code.

```
# this is a comment: addition
print(2+8)
```

```
print(2+8) # addition
```

10

## Built-in functions

- There are functions readily available for use in a notebook.
- There is no need to import these built-in functions from external sources.
- You can find the list of built-in functions in the following [link](#).

Built-in Functions			
<b>A</b> abs() aiter() all() anext() any() ascii()	<b>E</b> enumerate() eval() exec()	<b>L</b> len() list() locals()	<b>R</b> range() repr() reversed() round()
<b>B</b> bin() bool() breakpoint() bytearray() bytes()	<b>F</b> filter() float() format() frozenset()	<b>M</b> map() max() memoryview() min()	<b>S</b> set() setattr() slice() sorted() staticmethod() str() sum() super()
<b>C</b> callable() chr() classmethod() compile() complex()	<b>G</b> getattr() globals()	<b>N</b> next()	<b>T</b> tuple() type()
<b>D</b> delattr() dict() dir() divmod()	<b>H</b> hasattr() hash() help() hex()	<b>O</b> object() oct() open() ord()	<b>V</b> vars()
	<b>I</b> id() input() int() isinstance() issubclass() iter()	<b>P</b> pow() print() property()	<b>Z</b> zip()
			<b>_</b> <u>import_</u> ()

```
# absolute value function  
print(abs(-7))
```

7

```
# maximum of a list of numbers
```

[Skip to main content](#)

9

```
# minimum of a list of numbers  
print(min(3,-4,0,9,2))
```

-4

```
# sum of a list of numbers in a square bracket  
print(sum([3,5,2]))
```

10

```
# rounding to the nearest hundredth  
print(round(3.4678, 2))
```

3.47

```
# notebook only displays the value in the last line of the code cell  
5-1 # not displayed  
3+7 # displayed
```

10

```
# print() displays the outputs  
print(5-1) # displayed  
print(3+7) # displayed
```

4

10

## Indentation

[Skip to main content](#)

- Indentation refers to the spaces at the beginning of a code line.
- Python uses indentation to indicate a block of code.
- Do not use indentation unless you need it.
- Indentation refers to the spaces at the beginning of a code line.
- Python uses indentation to group the block of code.
- Only use indentation when necessary.

```
# ERROR: Indentation error
# There should not be space at the beginning
print(3+1)
    print(5+6)
```

## Import Modules/Packages/Libraries

- Only very commonly used, simple functions are available in a notebook by default.
- Python has a vast set of modules, packages and libraries where one can find numerous functions.
- For instance, the `mean()` function is not available by default in a notebook.

```
# ERROR: 'mean' is not defined
mean(4,8)
```

- The mean function can be imported from the `statistics` module or `numpy` library.
- You can import a package and call any function from it using the `package.method`.
- You can import only the function(s) you need from the package using the `from PACKAGE import METHOD`.
- Additionally, you can import a package and rename it for shorter usage by employing the `import PACKAGE as ABBREVIATION`.

```
import statistics      # import whole module
statistics.mean([4,8]) # call mean method from statistics module
```

```
from statistics import mean          # import only mean method  
mean([4,8])                         # call mean method
```

6

```
import statistics as st              # import whole statistics module as st  
st.mean([4,8])                      # call mean method from st module
```

6

```
import numpy as np                  # import numpy package as np  
np.mean([4,8])                     # call mean method from np package
```

6.0

- You can call mathematical constants and functions from the `math` module.
- Functions in modules are also referred to as methods.
- The built-in `dir()` and `help()` functions display the lists of constants (attributes) and functions (methods).

```
# displays all functions and constants in math module  
import math  
dir(math)
```

```
# displays all functions and constants in math module with explanation  
import math  
help(math)
```

```
import math  
dir(math)
```

```
['__doc__',
 '__file__',
 '__loader__',
 '__name__',
 '__package__',
 '__spec__',
 'acos',
 'acosh',
 'asin',
 'asinh',
 'atan',
 'atan2',
 'atanh',
 'cbrt',
 'ceil',
 'comb',
 'copysign',
 'cos',
 'cosh',
 'degrees',
 'dist',
 'e',
 'erf',
 'erfc',
 'exp',
 'exp2',
 'expm1',
 'fabs',
 'factorial',
 'floor',
 'fmod',
 'frexp',
 'fsum',
 'gamma',
 'gcd',
 'hypot',
 'inf',
 'isclose',
 'isfinite',
 'isinf',
 'isnan',
 'isqrt',
 'lcm',
 'ldexp',
 'lgamma',
 'log',
 'log10',
 'log1p',
 'log2',
 'modf',
 'nan',
 'nextafter',
 'perm',
```

[Skip to main content](#)

```
'prod',
'radians',
'remainder',
'sin',
'sinh',
'sqrt',
'tan',
'tanh',
'tau',
'trunc',
'ulp']
```

```
{ print(math.pi)
```

```
3.141592653589793
```

```
{ # ln(10)
print(math.log(10))    # natural log 10
```

```
2.302585092994046
```

- The `datetime` module contains methods that can be used for working with time and date

```
{ import datetime
print(datetime.datetime.now())      # date and time now
```

```
2024-02-02 23:50:34.368893
```

- You can use the `calendar` module to display a monthly calendar.

```
{ import calendar
print(calendar.month(1900, 1))      # import calender module
                                         # display Jan month of 1900
```

January 1900						
Mo	Tu	We	Th	Fr	Sa	Su
1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28
29	30	31				

- You can use the `random` module to generate random numbers or make random choices.

```
import random
random.randint(1,10) # choose a random integer between 1 and 10
```

3

```
random.choice(['a','b','c']) # choose a random element from a list
```

'c'

## Magic Methods

- The methods in `dir(math)` enclosed by double underscores `__` are referred to as magic functions.
- These functions operate in the backend and are not intended for direct user use.
- While it is still possible to use them, there is usually a more user-friendly alternative.
- For example, if you examine `dir(int)`, you will find the `__add__` method, which handles the addition operation internally, corresponding to the use of the `+` operator.

```
number = 3
print(number + 5)
print(number.__add__(5)) # same as 3+5
```

8

## Images

- You can import images from a website using the IPython library.
- Adjust the size of the image by utilizing width and height parameters.

```
from IPython.core.display import Image  
Image('https://www.python.org/static/community_logos/python-powered-h-100x130.png', width=100, height=130)
```



## Youtube Video

- You can import YouTube videos using the unique ID number.
- The ID number of a YouTube video can be found in the video's URL directly after the = sign
- ID number of <https://www.youtube.com/watch?v=rNgsRZ2C1Y> is `rNgsRZ2C1Y`

```
from IPython.lib.display import YouTubeVideo  
YouTubeVideo('rNgsRZ2C1Y', width=800, height=300)
```

## Google Colab features you may have missed



# Code Cell Questions

## Questions

Perform each of the following algebraic operations in a separate code cell:

- $20 - 3$
- $72 + 34$
- $46 \div 4$
- $4 \times 12$
- $5^3$

## Solution

▶ Show code cell content

## Question

Find the difference between the maximum and minimum of the following numbers:

65, 23, 1, 2, 3, 90, 99, 12, 34, 67 using the built-in *max()* and *min()* functions.

## Solution

▶ Show code cell content

## Question

Print the string 'Python' five times using string repetition."

[Skip to main content](#)

▶ Show code cell content

## Question

Print the mean, median, mode, and standard deviation of the following list of numbers: [12, 3, 12, 9, 7, 3, 25, 6], using the statistics module.

- You can refer to the available functions in the statistics module using `dir(statistics)` or `help(statistics)`.

## Solution

▶ Show code cell content

## Question

Print the square root of 75 using the `math`, `statistics` and `numpy` modules.

## Solution

▶ Show code cell content

# Colab Exercises

Please follow the steps below:

1. Create a folder in your Google Drive and rename it as `python_labs`.
2. Within this folder, create a Colab notebook and give it the name `firstname_lastname_lab_colab` using your own name.
3. Insert a `text` cell, enter *Biography*, and set it as a heading with level-1.
4. Enter *Albert Einstein* to the same text cell, and move it to become the first cell in the notebook.
5. Below that, insert another `text` cell, enter *Education*, and set it as a heading with level-2.
6. Include the following two items in this text cell as a bullet list, with degrees and years as sublists.
  - Federal polytechnic school (Dipl., 1900)
  - University of Zurich (PhD, 1905)

[Skip to main content](#)

8. Include the following items in this text cell as a numbered list.

- Barnard Medal (1920)
- Nobel Prize in Physics (1921)
- Matteucci Medal (1921)
- ForMemRS (1921)
- Copley Medal (1925)
- Gold Medal of the Royal Astronomical Society (1926)
- Max Planck Medal (1929)
- Member of the National Academy of Sciences (1942)
- Time Person of the Century (1999)

9. After the second item above, insert a horizontal line.

10. Change the color of the third item to red.

11. Emphasize the fourth item by making it bold.

12. Increase the font size of the seventh item to 20.

13. Italicize the eighth item.

14. Highlight the last one.

15. Create a new text cell and construct a table with two columns and nine rows, including *Award* and *Year* as the column headers.

16. Insert four code cells, perform one algebraic operation in each, and include a meaningful comment for clarity in each cell.

17. Insert a code cell and print 'Tesla' 10 times in a single line.

18. Insert a code cell and embed a YouTube video.

19. Insert a code cell and embed a picture from a website.

20. Insert a code cell and calculate the median of the numbers: 9, 34, 13, 90, 5.

**Hint:** The output of the text cells should be as follows:

## Biography

Albert Einstein

- Federal polytechnic school
  - Dipl.
  - 1900
- University of Zurich
  - PhD
  - 1905

## Awards

1. Barnard Medal (1920)
2. Nobel Prize in Physics (1921)
3. **Matteucci Medal (1921)**
4. **ForMemRS (1921)**
5. Copley Medal (1925)
6. Gold Medal of the Royal Astronomical Society (1926)

## 7. Max Planck Medal (1929)

8. *Member of the National Academy of Sciences (1942)*
9. **Time Person of the Century (1999)**

Award	Year
Barnard Medal	1920
Nobel Prize in Physics	1921
Matteucci Medal	1921
ForMemRS	1921
Copley Medal	1925
Gold Medal of the Royal Astronomical Society	1926
Max Planck Medal	1929
Member of the National Academy of Sciences	1942
Time Person of the Century	1999

## Chp-1: Variables

- Learning Objectives
  - ..
  - ..

A variable is a name that refers to a value, such as a number (e.g., 25) or a name (e.g., 'Michael').

- It functions like a container for storing data values.
- Variables allow you to store and manipulate data in your programs.
- To create a variable:
  1. Choose a name.
  2. Assign a value to it using the `=` operator.
- Example: `age = 25`
  - The variable's name is `age`.

— · — · — · — · —

[Skip to main content](#)

# Purpose

- Why do we need variables?

For the code below:

```
print(40)
print(40+1)
print(40**2)
print(40*7)
print(40-13)
```

- How many changes do you need to make to get a similar output for 20 instead of 40?
  - Answer: 5

```
print(20)      # 1st change
print(20+1)    # 2nd change
print(20**2)   # 3rd change
print(20*7)    # 4th change
print(20-13)   # 5th change
```

- In a lengthy program, you might need to make numerous changes, which can be time-consuming.
- There is a risk of forgetting to update some of them, leading to errors or incorrect output.
- Let's now write the same code, but this time using the variable *number*.

```
number = 40
print(number)
print(number+1)
print(number**2)
print(number*7)
print(number-13)
```

- If you use the variable `number`, then to obtain the output for 20, you only need to make one change, as follows:

```
number = 20      # 1st change
print(number)
print(number+1)
print(number**2)
print(number*7)
...  
...
```

[Skip to main content](#)

# `id()`

The `id()` function is a built-in function that returns the memory location of a variable.

- It can also be considered as the ID number of the variable.

```
age = 25  
print(id(age))
```

```
4300337000
```

```
state = 'Florida'  
print(id(state))
```

```
4356362800
```

## Values and Types

In Python, the most commonly used basic data types are integers, floats, strings, and booleans.

### 1. Integers

- These are numbers without a decimal point, such as  $-3, -2, -1, 0, 1, 2, 3, \dots$ .
- Their class is called *int*.
- Examples: 5, 9, 123.

### 2. Floats

- These are decimal numbers, including those with a decimal point, such as 4.0.
- Their class is called *float*.
- Examples: 4.89, 67.98, 5.0

### 3. Strings

- These are ordered sequences of characters.
- They are enclosed in quotes.
- Their class is called *str*.

[Skip to main content](#)

## 4. Booleans

- There are only two boolean values: `True` and `False`.
- They represent the truth values of boolean expressions.
- They are keywords in Python and cannot be used as variable names.
- Their class is called `bool`.
- The boolean `True` and the string `'True'` are different.
- The boolean `False` and the string `'False'` are different.

In Python, there's no need to explicitly specify the type of a variable.

- The type is determined automatically based on the assigned value.

## type()

The `type()` function is a built-in function that returns the data type of a variable or value.

```
# type of 2
print(type(2))
```

```
<class 'int'>
```

```
# type of 2.97
print(type(2.97))
```

```
<class 'float'>
```

```
# type of 2.0
print(type(2.0))
```

```
<class 'float'>
```

```
# type of 'Florida'
```

[Skip to main content](#)

```
<class 'str'>
```

```
# type of boolean True  
print(type(True))
```

```
<class 'bool'>
```

```
# type of boolean False  
print(type(False))
```

```
<class 'bool'>
```

```
# type of string 'True'  
print(type('True'))
```

```
<class 'str'>
```

```
# type of string '2'  
print(type('2'))
```

```
<class 'str'>
```

Warning:

- `2` is an integer, making it a number
- `'2'` is a string; it represents the character 2 and is not considered a number.

## Variable names

Variable names can contain lowercase and uppercase characters, digits, and underscores with the following properties:

[Skip to main content](#)

- Variable names are case-sensitive.
- A variable name cannot start with a digit.
- The underscore character `_` can be used in a name as a substitute for a space.
- Variable names cannot be any of the keywords (reserved words) listed below:

```
# keywords in Python
help('keywords')
```

Here is a list of the Python keywords. Enter any keyword to get more help.

False	class	from	or
None	continue	global	pass
True	def	if	raise
and	del	import	return
as	elif	in	try
assert	else	is	while
async	except	lambda	with
await	finally	nonlocal	yield
break	for	not	

```
# case sensitive
name = 'Jim'
Name = 'Joe'
```

- The value of the `Name` variable is different from that of the `name` variable.

```
print(name)
```

Jim

```
print(Name)
```

Joe

```
# ERROR: 5x is not a valid variable name
5x=9
```

[Skip to main content](#)

```
# ERROR: keyword error  
False = 5
```

```
# ERROR: symbol error  
@fun=4
```

**Remark:** Function names can be used as a variable name but function properties will be lost.

- Therefore, it is not recommended to use the function names as variable names

```
sum = 6  
sum([7,8]) # ERROR: sum can not do addition
```

- In the first line of the code above, *sum* becomes a variable and its value is 6.
- In the second line, *sum* is attempted to be used as a function, but it is no longer a function, resulting in an error message.

## Reassignment

- The value of a variable can be changed by assigning it a new value.

```
age = 50  
print(age)  
age = 25      # The value of the age variable has been changed.  
print(age)  
age = 'ten'   # The type can also be changed.  
print(age)
```

```
50  
25  
ten
```

## Multiple assignment

Multiple variables can be created in a single line.

[Skip to main content](#)

```
# a=1, b=10, c=100  
a, b, c = 1, 10, 100
```

```
print(a)  
print(b)  
print(c)
```

```
1  
10  
100
```

- In the following statement, the value 7 is assigned to both variables x and y.
- It's a shorthand way of assigning the same value to multiple variables in a single line.

```
x = y = 7
```

```
print(x)  
print(y)
```

```
7  
7
```

## Print variables and constants

Variable values and constant strings can be displayed using a single *print()* function. There are several ways to do this.

- One way is to separate variables and constant strings with commas.
- By default, there will be a space in the output between each comma-separated item.

```
name, age = 'Michael', 25
```

```
print('My name is', name)
```

[Skip to main content](#)

My name is Michael

```
print('I am', age, 'years old.')
```

I am 25 years old.

```
print('My name is', name, '.I am', age, 'years old.')
```

My name is Michael .I am 25 years old.

- You can do algebraic operations in the print function

```
print(2024-age)
```

1999

## Changing the type of a variable

- int(): converts appropriate floats and strings into integers
- float(): converts appropriate integers and strings into floats
- str(): converts appropriate integers and floats into strings

```
# float to int  
print(int(3.45))
```

3

```
# float to int  
print(int(-3.2))
```

[Skip to main content](#)

-3

```
# str to int  
print(int('3'))
```

3

```
# ERROR: string '3.59' to int  
print(int('3.59'))
```

```
# int to float  
print(float(1))
```

1.0

```
# str to float  
print(float('3.5'))
```

3.5

```
# int to str  
print(str(1))
```

1

```
# float to str  
print(str(3.5))
```

3.5

[Skip to main content](#)

```
age = 25  
print('I am '+str(age)+' years old.')
```

I am 25 years old.

## String Operations

### Concatenation

- Combining or joining two or more strings into a single string by using the `+` operator

```
first_name = 'Michael'  
last_name = 'Jordan'  
age = 25
```

- In this example, the `+` operator concatenates (joins) the value of `first_name` and `last_name` variables to create a new string stored in the variable `name`.

```
name = first_name + last_name  
print(name)
```

MichaelJordan

- Use the string `' '` (one space) to add a space between the first and last names.

```
name = first_name + ' ' + last_name  
print(name)
```

Michael Jordan

- constant string + variable

My name is Michael Jordan

- To perform concatenation, conversion should be done in the following code.

```
print('I am ' + str(age) + ' years old.')
```

I am 25 years old.

- + can not be used between numbers and strings

```
name, age = 'Michael', 25  
# ERROR: str + int  
print(name+age)
```

## Repetition

- Repeating a string using the \* operator.
- It takes the form of integer\*string or string\*integer.
- String repetition cannot be performed using floats, even with 4.0
- In this example, the \* operator repeats the string 'U' four times.

```
print('U'*4)
```

UUUU

- In this example, the \* operator repeats the string 'Joe' five times.

```
print('Joe'*5)
```

JoeJoeJoeJoeJoe

[Skip to main content](#)

- Since 4.0 is not an integer, you will get an error message in the following code.

```
# ERROR: 4.0 must be 4
print('U'*4.0)
```

- The following is a triangle constructed using the \$ character.

```
print('$')          # one   $ sign
print('$'*2)        # two   $ signs
print('$'*3)        # three $ signs
print('$'*4)        # four   $ signs
print('$'*5)        # five   $ signs
```

```
$
$$
$$$
$$$$
$$$$$
```

- The following is a triangle constructed using the space and \$ characters.

```
# triangle by using the space and $ characters
print(' '*4 + '$')      # four spaces and one $ sign
print(' '*3 + '$'*2)    # three spaces and two $ signs
print(' '*2 + '$'*3)    # two spaces and three $ signs
print(' '*1 + '$'*4)    # one space and four $ signs
print('$'*5)            #           five $ signs
```

```
$
$$
$$$
$$$$
$$$$$
```

## Variables Debugging

- Each of the following short code contains one or more bugs.
- Please identify and correct these bugs.
- Provide an explanation for your answer.

[Skip to main content](#)

## Question-1

```
state = 'NY'  
int(state)
```

### Solution

string `state` can not be converted to an integer.

## Question-2

```
991_number = 'emergency'
```

### Solution

variable names can not start with a digit

## Question-3

```
float('HI')
```

### Solution

string `HI` can not be converted to a float.

## Question-4

```
A = 3  
print(a + 5)
```

[Skip to main content](#)

### Solution

`a` is not defined because capital `A` is 3

## Question-5

```
True = 5  
print(True)
```

### Solution

Keywords can not be used as a variable names. `True` is a keyword.

## Question-6

```
print(int('5.0'))
```

### Solution

`5.0` is not an integer

## Question-7

```
age = 25  
print('I am'+age+'years old.')
```

### Solution

- `I am` is a string and `age` is an integer.
- '+' can not be used to concatenate numbers and strings

```
print = 25
print('Hello')
```

### 🔔 Solution

- In the first line, *print* becomes a variable, and its value is 25, so it loses its function property.



## Variables Output

- Find the output of the following code.
- Please don't run the code before giving your answer.

### Question-1

```
print(type(5))
```

► Show code cell output

### Question-2

```
print(type(5.5))
```

► Show code cell output

### Question-3

```
print(type(5.0))
```

► Show code cell output

### Question-4

[Skip to main content](#)

```
print(type('5'))
```

► Show code cell output

## Question-5

```
x = 5  
y = 'apple'  
print('I have', 20-x, y, 's.')
```

► Show code cell output

## Question-6

```
age = 25  
print('I am' + age +'years old.')
```

► Show code cell output

## Question-7

```
age = 25  
print('I am' +str(age)+ 'years old.')
```

► Show code cell output

## Variables Code

- Please solve the following questions using Python code.

## Question-1

Create a variable named `course` and assign the value `math` to it.

**Solution**

[Skip to main content](#)

▶ Show code cell content

## Question-2

Create a variable named `weight` and assign the value `180.4` to it.

### Solution

▶ Show code cell content

## Question-3

Create a variable named `number_of_apples` and assign the value `45` to it.

### Solution

▶ Show code cell content

## Question-4

Write the code to convert the integer `1` to the string `'1'`

### Solution

▶ Show code cell content

## Question-5

- Create a variable named `x` and assign the value `7` to it.
- Create a variable named `y` and assign the value `3` to it.
- Display the sum, difference, and product of `7` and `3` using the variables `x` and `y`.

### Solution

▶ Show code cell content

[Skip to main content](#)

## Question-6

- Create a variable named `x` and assign the value 7 to it.
- Create a variable named `y` and assign the value 3 to it.
- Create a variable named `z`, assign the value of `x * y` to it.
- Display `z`.

### Solution

 ▶ Show code cell content

## Question-7

- Create a variable named `width` and assign the value `10` to it.
- Create a variable named `length` and assign the value `50` to it.
  - These variables represent the sides of a rectangle.
- Create a variable named `perimeter` and assign the value of the rectangle's perimeter to it.
- Create a variable named `area` and assign the value of the rectangle's area to it.
- Display the following statements by using the variables `width`, `length`, `perimeter` and `area`.
  - The width of the garden is 10 ft.
  - The length of the garden is 50 ft.
  - The area of the rectangle is 500 square ft, indicating a large size.
  - The perimeter of the rectangle measures 120 ft, indicating that a considerable amount of fencing is required.
- Please avoid using the numbers 10 and 50 in your code, and remember to include appropriate punctuation.

### Solution

 ▶ Show code cell content

## Question-8

- Create a variable named `x` and assign the value `3` to it. (integer type)

[Skip to main content](#)

- Without directly using the number 3 in your code (use variable x) print 3333 using three different code
- Assign 5 to x and verify if you obtain 5555 with the same code.

## Solution

▶ Show code cell content

# Variables Exercises

## Question-1

By using the variables x and y given below, print 353535 using 4 different pieces of code.

x, y = 3, 5

## Question-2

Print the letter U using the character +.

- The vertical left and right parts of U have five + characters each.

horizontal = 3

- The variable horizontal represents the number of + characters with one space between them in the bottom horizontal part.

[Skip to main content](#)

## Examples

horizontal = 3	horizontal = 7	horizontal = 10	horizontal = 15
+ + + + + + + + + + +	+ + + + + + + + + + + + +	+ + + + + + + + + + + + + + +	+ +

## Question-3

Print the statement "I went to Italy in 2015." using the country and year variables below.

```
country = 'Italy'  
year = 2015
```

*Warning: There is no space before the period.*

## Question-4

Create a variable named `r` and assign the value of 5 to it.

- Print the area and perimeter of the circle with a radius equal to `r`.
- Round them to the nearest hundredth.
- Hint:
  - Perimeter =  $2\pi r$ , Area =  $\pi r^2$
  - Import  $\pi$  from a module.
- Output:
  - Perimeter: 31.42
  - Area : 78.54

[Skip to main content](#)

Print the following using the character `*`, spaces and the variable `n=5`.

```
*****
 * *****
  * ****
   * ****
    * ****
     * ****
      * ****
```

## Chp-2: Input and Output

- Learning Objectives
  - ..
  - ..

### input()

The `input()` function is a built-in function used to obtain data or information from the user.

- It returns a string.
- To receive a number from the user, you'll need to convert the output of the `input()` function to an integer or float.
- You can include a message as a string to provide directions to the user.
- Upon running the code, your message will be displayed, and a box will prompt the user for input.
- After entering the input, the user should press the enter key.

In the following code, the user is prompted with the message `Enter your birth year:`.

```
input('Enter your birth year: ')
```

Two important points in the above code:

1. Even though the user enters a number, the `input()` function returns a string.

[Skip to main content](#)

2. We must assign the value given by the user to a variable to store and use it.

- In the provided code, as no variable is used, there's no way to access the given birth year in subsequent lines.

In the following code, we will once again ask for the user's birth year, but this time we will assign it to a variable.

- This way, we will be able to access the birth year through the variable `birth_year` in any cell.
- Note that the type of the `birth_year` variable is *string* because the `input()` function returns any entered value as a string.

```
birth_year = input('Enter your birth year: ')
print('birth_year type:', type(birth_year))
```

## Output

Enter your birth year: 2000  
birth\_year type: <class 'str'>

- If you intend to perform algebraic operations, such as calculating age, using the `birth_year` variable, you'll need to convert it to a numerical type.
- Otherwise, an error message will be generated.

```
birth_year = input('Enter your birth year: ')
age = 2024 - birth_year # ERROR: integer 2024 - string birth_year
```

- To avoid this, convert `birth_year` to an integer. This can be done in a couple of different ways.
- In the second line of the following code, the integer value of `birth_year` is used in subtraction.

```
birth_year = input('Enter your birth year: ')
age = 2024 - int(birth_year)

print('Your age is', age)
print('birth_year type:', type(birth_year))
```

## Output

[Skip to main content](#)

Your age is 24

birth\_year type: <class 'str'>

- In the above code, the type of `birth_year` was not changed; it remains a string because we did not assign a new value to it.
- This can be accomplished in a concise manner at the very beginning of the code.

```
birth_year = input('Enter your birth year: ')
birth_year = int(birth_year)    # assign a new value to the birth_year variable

age = 2024 - birth_year

print('Your age is', age)
print('birth_year type:', type(birth_year))
```

## Output

Enter your birth year: 2000

Your age is 24

birth\_year type: <class 'int'>

- There is a shortcut for performing this conversion.
- Upon receiving input from the user, we can immediately convert that value to an integer.
- In the following code, the `input()` function returns a string, and the `int()` function converts this string to an integer.

```
birth_year = int(input('Enter your birth year: '))

age = 2024 - birth_year

print('Your age is', age)
print('Birth year type:', type(birth_year))
```

## Output

Enter your birth year: 2000

Your age is 24

Birth year type: <class 'int'>

## Receipt Example

[Skip to main content](#)

- In the following example, the user enters the quantity of hamburgers and sodas, and the final receipt, including tax and tip, is printed.
  - The price of a hamburger is \$5.
  - The price of a coke is \$2.
  - The tip is 15%.
  - The tax is 10%.
- Possible improvements:
  - You can also include the time and date by using the datetime module.
  - Consider rounding the tax, tip, and total amounts.

```

hamburger = int(input('Number of hamburgers:'))
soda = int(input('Number of sodas:'))

subtotal = hamburger*5+soda*2
tip = subtotal*0.15
tax = subtotal*0.10
total = subtotal+tip+tax

print('Hamburger:',hamburger,'x 5= ',hamburger*5)
print('Soda      :',soda,'x 2= ',soda*2)
print('Tip       :          ',tip)
print('Tax       :          ',tax)
print('Total     :          ',total)

```

## Output

Number of hamburgers:10

Number of sodas :20

Hamburger: 10 x 5= \* 50

Soda : 20 x 2= 40

Tip : 13.5

Tax : 9.0

Total : 112.5

## Whitespaces

The following whitespace characters are frequently used in print statements for spacing.

[Skip to main content](#)

- Moves to the next line.
- `\t`: tab
  - Inserts a tabulation
  - Inserts spaces up to the next tab stop, which occurs every 8th character.
- `\b`: backspace
  - Deletes the character to the left.
- `\r`: carriage return
  - Moves to the beginning of the line.
  - In Jupyter notebook, it does not delete any characters.
  - In Google Colab, it deletes all characters.

```
print('A\nB') # after A it moves to the next line
```

A  
B

```
print('A      B')    # 7 spaces
print('A'+ ' '*7+'B') # repetition of the string ' ' (one space) seven times
print('A\tB')        # tab
```

A B  
A B  
A B

```
print('AA      B')    # 6 spaces
print('AA'+ ' '*6+'B') # repetition of the string ' ' (one space) six times
print('AA\tB')        # tab
```

AA B  
AA B  
AA B

```
print('AAA      B')    # 5 spaces
print('AAA'+ ' '*5+'B') # repetition of the string ' ' (one space) five times
```

[Skip to main content](#)

```
AAA      B  
AAA      B  
AAA      B
```

```
print('ABC\bD') # C is deleted by '\b'
```

```
ABCD
```

```
print('ABC\b\bD') # C and B are deleted by two '\b' s
```

```
ABCD
```

For Jupyter Notebook:

- In the following code, after the character 'C', the carriage return `\r` moves the cursor to the beginning of the line, and 'D' overwrites 'A'.

```
print('ABC\rD') # carriage return
```

- Output: DBC
- **Warning:** In Google Colab, the output of the provided code is D.

## print()

The `print()` function is a built-in function that displays output on the screen.

- It has two significant parameters: `sep` and `end`.

## sep parameter

- It is the separator parameter.
- It determines what to insert between the comma-separated values in a print function.

[Skip to main content](#)

- sep values are strings

```
name = 'Tom'
age = 25
print('A', age, 'B', name)           # by default there is one space between each value
```

A 25 B Tom

```
name = 'Tom'
print('A', age, 'B', name, sep='--') # values are separated by one '-' (dash)
```

A--B--Tom

```
name = 'Tom'
print('A', age, 'B', name, sep='***') # values are separated by three '*'s (asterisk)
```

A\*\*\*25\*\*\*B\*\*\*Tom

## end parameter

- It determines what to print at the end of the output.
- The default value of end parameter is the new line: '\n'.
- end values are strings

### Example

```
print('A') # end='\n' by default, after printing A it moves to next line
print('B') # end='\n' by default, after printing B it moves to next line
print('C') # end='\n' by default, after printing C it moves to next line
print('D') # end='\n' by default, after printing D it moves to next line
```

A  
B  
C

[Skip to main content](#)

## Example

```
print('A', end='--') # end='--' , after printing A it prints '--'
print('B')          # end='\n' by default, after printing B it moves to next line
print('C')          # end='\n' by default, after printing C it moves to next line
print('D')          # end='\n' by default, after printing D it moves to next line
```

A--B  
C  
D

## Example

```
print('A', end='--') # end='--' , after printing A it prints --
print('B', end='+') # end='+' , after printing B it prints +
print('C')          # end='\n' by default, after printing C it moves to next line
print('D')          # end='\n' by default, after printing D it moves to next line
```

A--B+C  
D

## Example

```
print('A')          # end='\n' by default, after printing A it moves to next line
print('B', end='+') # end='+' , after printing B it prints +
print('C', end='?') # end='+' , after printing C it prints ?
print('D')          # end='\n' by default, after printing D it moves to next line
```

A  
B+C?D

## Examples

### Moving O (right)

I use the `print()` function along with the letter 'O' (the backspace '\b') and the `open()` function from the

[Skip to main content](#)

- Each `print('\b O', end='')` statement performs four actions:
  1. '\b' deletes the 'O' that was printed before, moving the cursor one position to the left.
  2. Prints a space.
  3. Prints 'O'.
  4. Since the end parameter is set to an empty string, it does not move to the next line.

```
import time
print('O', end='')
time.sleep(1)
print('\b O', end='')      # Delete the 'O', print a space, then print 'O'.
time.sleep(1)
print('\b O', end='')
```

## Moving O (left)

Use the `print()` function along with the letter 'O', the backspace '\b', and the `sleep()` function from the `time` module to create a left-moving 'O'!

- Each `print('\b'*2 +'O', end='')` statement performs four actions:
  1. The first '\b' deletes the 'O' that was printed before, moving the cursor one position to the left.
  2. The second '\b' moves the cursor one position to the left.
  3. Prints 'O'.
  4. Since the end parameter is set to an empty string, the cursor does not move to the next line after printing the 'O'.

```
import time
print(' '*5 + 'O', end='')      # Print 'O' after 5 spaces.
time.sleep(1)
print('\b'*2 +'O', end='')      # Delete the 'O', print a space, then print 'O'.
time.sleep(1)
print('\b'*2 +'O', end='')
```

[Skip to main content](#)

```
time.sleep(1)
print('\b'*2 + '0', end='')
time.sleep(1)
print('\b'*2 + '0', end='')
time.sleep(1)
print('\b'*2 + '0', end='')
```

## Input and Output Debugging

- Each of the following short code contains one or more bugs.
- Please identify and correct these bugs.
- Provide an explanation for your answer.

### Question-1

```
print 'Hello'
```

#### Solution

Parentheses are missing.

### Question-2

```
print('Hello)
```

#### Solution

Single quote on the right of Hello is missing.

### Question-3

```
print('Hello'))
```

[Skip to main content](#)

### Solution

There is an extra parenthesis at the end.

## Question-4

```
print = 3  
print('England')
```

### Solution

- In the first line print becomes a variable and it's value is 3.
- In the second line print is tried to used as a function but it is not a function anymore.

## Question-5

```
x = input(Enter your birthyear:)  
print(x)
```

### Solution

Single quotes of the string inside the input function are missing.

## Question-6

```
x = input('Enter your birthyear: ')  
print(f'Your age is {2022-x}')
```

### Solution

Type of x is string so subtraction cannot be done.

## Question-7

```
print('A',2,3,'B', separator='++')
```

### Solution

The name of the parameter is `sep` (not `separator`).

## Question-8

```
first_name = input('Enter your first name: ')
print(first_name)
```

### Solution

Variable names (`first name`) cannot have a space.

## Input and Output Output

- Find the output of the following code.
- Please don't run the code before giving your answer.

## Question-1

```
name = 'Liz'
print('name')
```

► Show code cell output

## Question-2

```
number = 49
```

[Skip to main content](#)

▶ Show code cell output

## Question-3

```
print('Michael'+'Jordan')
```

▶ Show code cell output

## Question-4

```
print('Michael'+' '+'Jordan')
```

▶ Show code cell output

## Question-5

```
print('Michael'+' '+'Jordan')
```

▶ Show code cell output

## Question-6

```
print('Michael', 'Jordan')
```

▶ Show code cell output

## Question-7

```
age = 25  
print('I am' +'age'+ 'years old.')
```

▶ Show code cell output

[Skip to main content](#)

```
age = 25  
print('I am' +str(age) +'years old.')
```

► Show code cell output

## Question-9

```
x = 5  
y = 'apple'  
print('I have', 20-x, y, 's.')
```

► Show code cell output

## Question-10

```
number = input('What is your favorite number: ')  
print('number')
```

► Solution

number



## Question-11

```
print('Hel\nlo')
```

► Show code cell output

## Question-12

```
print('Hel\tlo')
```

► Show code cell output

[Skip to main content](#)

## Question-13

```
print('He\blo')
```

► Show code cell output

## Question-14

```
print('TEX\b\AS')
```

► Show code cell output

## Question-15

```
print('CA\nLIFO\tRN\bIA')
```

► Show code cell output

## Question-16

```
x = '5'  
y = '7'  
print(x+y)
```

► Show code cell output

## Question-17

```
print('H',2,3,'BYE', sep='++')
```

► Show code cell output

## Question-18

[Skip to main content](#)

▶ Show code cell output

## Question-19

```
print('A')
print('B', end='--')
print('C')
print('D', end='??')
print('E')
```

▶ Show code cell output

## Question-20

```
print('A', 'A', 'A', sep='8', end='K')
print('B', 'B', sep='--', end='L')
print('C')
print('D')
```

▶ Show code cell output

## Input and Output Code

- Please solve the following questions using Python code.

## Question-1

Display the first 3 characters of your first name by using the character \*.

### Solution

▶ Show code cell content

## Question-2

- Use the given variables to print the following statement:

My name is Michael. I am from Germany. I am 25 years old.

[Skip to main content](#)

- name = 'Michael'
- age = 25
- country = 'Germany'

## Solution

▶ Show code cell content

## Question-3

Write a program that prompts the user for 4 numbers using 4 input functions.

- Find the sum of these numbers and assign it to a variable.
- Find the product of these numbers and assign it to a variable.
- Print the sum and product of these numbers on two separate lines using a single print function.

### Solution

```
num1 = float(input('Number1: '))
num2 = float(input('Number2: '))
num3 = float(input('Number3: '))
num4 = float(input('Number4: '))

total = num1 + num2 + num3 + num4
product = num1 * num2 * num3 * num4

print('Sum: ', total, '\nProduct:', product)
```

## Question-4

- Use the given variables to print **Bill- -Gates** by using five different codes.
  - first\_name = 'Bill'
  - last\_name = 'Gates'

## Solution

▶ Show code cell content

[Skip to main content](#)

## Question-5

Write a program which prompts the user for an integer.

- Find the square of the given number.
- Print the following statement:
  - $\text{square}(\text{number}) = \text{square of the number}$
  - Example: If the given number is 3 then the output should be  $\text{square}(3)=9$
  - Example: If the given number is 5 then the output should be  $\text{square}(5)=25$

### Solution

```
number = int(input('Enter an integer: '))
square = number**2
print('square(', number, ')=', square, sep='')
```

## Input and Output Exercises

### Question-1

Write a single print statement that produces the following output:

- A
- B
- CD
- E

### Question-2

Use 2 *print()* functions, and the strings 'A', 'B', 'C', 'D' to generate the following output. A—B—C—D

### Question-3

[Skip to main content](#)

## Question-4

Write a program that prompts the user for 3 numbers using 3 *input()* functions.

- Find the harmonic mean of these numbers using the formula:  $H(x, y, z) = \frac{3}{\frac{1}{x} + \frac{1}{y} + \frac{1}{z}}$
- Round the harmonic mean to the nearest hundredth and print it in the following format:  
 $H(x, y, z) = \text{rounded harmonic mean.}$
- Sample Output:

Enter x: 2

Enter y: 3

Enter z: 4

$H(2.0, 3.0, 4.0) = 2.77$

## Question-5

Write a program that prompts the user for a positive integer.

- Print that integer.
- On the second line, print one more than the given number two times separated by a dash.
- On the third line, print two more than the given number three times separated by dashes.
- On the fourth line, print three more than the given number four times separated by dashes.
- On the fifth line, print four more than the given number five times separated by dashes.

Sample Output:

Please enter an integer: 5

5

6-6

7-7-7

8-8-8-8

9-9-9-9-9

## Question-6

[Skip to main content](#)

- Print the given word as many times as the given number.

## Question-7: Std

Write a program that prompts the user for 3 numbers using 3 *input()* functions.

- Find the mean of these three numbers.
- Subtract the mean from each number.
- Square each of these differences.
- Find the mean of these squares.
- Take the square root of this mean and round it to the nearest hundredth.

*Remark:* This is the standard deviation of the given numbers, which provides information about how the given numbers are spread out around the mean.

## Question-8: Countdown

Write a program that prints the numbers starting from 5 down to 1 and displays 'START' after that.

- Use the *sleep()* function from the *time* module to add a 1-second waiting time between each number for observation.
- After printing a number, pause for 1 second, then delete it before printing the next number.
- The output of your program should resemble a countdown.

## Question-9: Arrow

Write a program that prints the moving arrow '—>'.

- Print the arrow, then add a one-second waiting time, then delete it.
- Print the arrow again with a single space in front of it, then add a one-second waiting time, then delete it.
- Repeat this procedure 10 times by increasing the space one more in each step.
- The output of your program should resemble a moving arrow to the right.

# Chp-3: Numbers

- Learning Objectives

- ..
- ..

## Number Types

In this chapter, three different types of numbers in Python will be covered, but integers and floats will be the primary types used in this book. *If the sections on Floats, Complex Numbers, and Scientific Notation seem too technical for you, feel free to skip them.*

### 1. Integers: ..., -2, -1, 0, 1, 2, ...

- Type: int
- In Python, underscores (rather than commas) are used to separate large numbers into groups of three digits.

---

### 2. Float : 4.5, 3.0

- Decimal numbers (They have decimal point).
- 4.0 is a float
- Type: float
- Float values are approximately stored, but they are close enough to their real values to maintain practical accuracy.

---

### 3. Complex Numbers: $3 + 4j$

- Real numbers with imaginary parts.
- In mathematics, the symbol  $i$  is used to represent the imaginary unit.
- In Python, the symbol  $j$  is used to represent the imaginary unit.
- The built-in `complex()` function is used to create complex numbers.
- Type: complex

```
# type of 5 is integer
print(type(5))
```

[Skip to main content](#)

```
<class 'int'>
```

```
# type of -5 is integer  
print(type(-5))
```

```
<class 'int'>
```

```
# type of 12.89 is float  
print(type(12.89))
```

```
<class 'float'>
```

```
# type of 4.0 is float  
print(type(4.0))
```

```
<class 'float'>
```

## Floats

- Python does not store the value 0.1 exactly.
- The following code reveals that there are non-zero numbers in the decimal tail, which are not expected to be there.
- The built-in `format()` function can be used to display more decimal places.

```
# 30 digits after decimal point  
print(format(0.1, '.30f'))
```

```
0.100000000000000005551115123126
```

- 0.5 is stored exactly in Python.

[Skip to main content](#)

```
# 30 digits after decimal point
print(format(0.5, '.30f'))
```

0.50000000000000000000000000000000

- 0.375 is stored exactly in Python.

```
# 30 digits after decimal point
print(format(0.375, '.30f'))
```

0.37500000000000000000000000000000

- This is because numbers, even decimal ones, are stored in base two.
- If you can represent a decimal number in base two, it will be stored exactly.
- However, some numbers, like 0.1, cannot be accurately represented in base two.
- Examples
  - If you try to write the 0.375 in base two, you get:
    - $0.375 = \frac{3}{8} = \frac{1}{4} + \frac{1}{8}$
    - Hence 0.375 in base two is 0.011 and stored exactly.
  - If you attempt a similar computation for 0.1, you will encounter an infinite series.
    - $0.1 = \frac{1}{16} + \frac{1}{32} + \frac{1}{256} + \frac{1}{512} + \dots$
    - Hence 0.1 is 0.000110011... in base two.

## Complex Numbers

- Use the built-in `complex()` function to create complex numbers.
- It takes two parameters: the real part and the imaginary part.
- With complex numbers, you can perform algebraic operations and find conjugates.

```
# real part = 2, imaginary part = 3
print(complex(2,3))
```

[Skip to main content](#)

$(2+3j)$

```
z = complex(2,3)
```

```
# real part of z  
print(z.real)
```

2.0

```
# imaginary part of z  
print(z.imag)
```

3.0

```
# conjugate of z  
print(z.conjugate())
```

$(2-3j)$

```
t = complex(5,7)  
print(t)
```

$(5+7j)$

```
# addition  
print(z + t)
```

$(7+10j)$

[Skip to main content](#)

```
# subtraction  
print(z-t)
```

(-3-4j)

## Scientific Notation

It is used to represent very large or very small numbers in a more compact form.

- In scientific notation, a number is written in the form of:
  - (a number between 1 and 10) x e(a power of 10)
    - Example:  $12345 = 1.2345 \times 10^4 = 1.2345e + 04$
    - Example:  $0.00123 = 1.23 \times 10^{-3} = 1.23e - 03$
- In scientific notation,
  - e+04 means  $10^4$
  - e-03 means  $10^{-3}$
- Instead of e, you can also use E for scientific notation.
- The `format()` function can be used to represent a number using scientific notation.

```
# Scientific notation e  
# The number 3 represents the number of digits up to which the given number is to be ro  
print(format(12645, '10.3e'))
```

1.264e+04

```
# round 1.2645 to the nearest tenth  
print(format(12645, '10.1e'))
```

1.3e+04

```
# You can also use E instead of e  
print(format(12645, '10.1E'))
```

[Skip to main content](#)

1.3E+04

## Large Numbers

- 1, 234, 578 is a large number, and commas are used to facilitate a better understanding of this number.
- In Python, underscores (\_) are used instead of commas due to the special functionalities associated with commas in Python.

```
print(1_234_578)
```

1234578

In the following code, comma-separated three values are printed.

- Since the sep parameter is by default one space, these three values are displayed with one space between them.

```
print(1,234,578)
```

1 234 578

## Operations on numbers

The following operations are commonly used in Python:

[Skip to main content](#)

Symbol	Operation
+	addition
-	subtraction
*	multiplication
**	exponent
/	division
//	divide and floor (integer division)
%	remainder

- Division in Python always returns a float, even in cases where the result is a whole number.
  - $12/4 = 3.0$
- Integer division rounds down to the nearest smaller integer.
  - Example:
    - $13/5 = 2.6$
    - $13//5 = 2$
  - Example:
    - $-12/5 = -2.4$
    - $-12//5 = -3$
- The remainder operation `%` returns the remainder of a division.
  - Example:
    - When we divide 13 by 5, the quotient is 2, and the remainder is 3.
    - $13\%5=3$

```
# addition
print(5+3)
```

```
# subtraction  
print(5-3)
```

2

```
# multiplication  
print(5*3)
```

15

```
# division  
print(13/5)
```

2.6

```
# exponent  
# 2 to the third power  
print(2**3)
```

8

```
# integer division  
print(13//5)
```

2

```
# remainder  
# when we divide 13 by 5, the remainder is 3.  
print(13%5)
```

3

[Skip to main content](#)

```
# power = 1/2 means square root  
print(49**1/2))
```

7.0

```
# square root of negative numbers are complex numbers.  
# square root of -1  
print((-1)**1/2))
```

(6.123233995736766e-17+1j)

- In the code above we expect to have only  $1j$  which is  $j$ .
- There is a real part in this complex number, which is supposed to be 0.
- The real part  $6.123233995736766e - 17$  is in scientific notation, representing  $6.123233995736766 \times 10^{-17}$
- This is a very small number close to 0.
- We have this small number as the real part because floats are stored approximately.

## Conversions

- The built-in `int()` function is used to convert floats and suitable strings to integers.
- The built-in `float()` function is used to convert integers and suitable strings to floats.
- The built-in `str()` function is used to convert integers and floats to strings.

```
# convert positive float x to integer y  
x = 7.56  
y = int(x)  
print('Type:', type(y))  
print('y =', y)
```

Type: <class 'int'>  
y = 7

```
# convert negative float x to integer y
x = -7.56
y = int(x)
print('Type:', type(y))
print('y =', y)
```

Type: <class 'int'>  
y = -7

```
# convert string x to integer
x = '7'
y = int(x)
print('Type:', type(y))
print('y =', y)
```

Type: <class 'int'>  
y = 7

```
# ERROR: 'Tom' cannot be converted to an integer.
int('Tom')
```

```
# convert integer x to float y
x = 7
y = float(x)
print('Type:', type(y))
print('y =', y)
```

Type: <class 'float'>  
y = 7.0

```
# convert string x to float y
x = '7.53'
y = float(x)
print('Type:', type(y))
print('y =', y)
```

Type: <class 'float'>

[Skip to main content](#)

- You cannot convert decimal numbers in string type to an integer.

```
# ERROR: string '7.63' cannot be converted to an integer.  
int('7.53')
```

- Extra spaces on the left or right do not affect the conversion process.

```
print(int(' 234 '))
```

234

## Precedence (PEMDAS)

The operation precedence of the operations in Python follows the following order:

- Parenthesis
- Exponents
- Multiplication, Division, Integer Division, Remainder
- Addition, Subtraction

```
# first multiplication, then addition  
print(2+5*3)
```

17

```
# first remainder, then addition  
print(2+5%3)
```

4

```
# first Integer Division, then addition  
print(2+5//3)
```

[Skip to main content](#)

3

```
# first exponent, then addition
print(2+5**3)
```

127

## Abbreviated operators

- In mathematics, the expression  $x = x + 1$  represents an equation. By subtracting  $x$  from both sides, the equation can be solved.
- In programming languages,  $x = x + 1$  is an assignment, not an equation. Remember that `=` is an assignment operator.
- In the assignment  $x = x + 1$ , the following steps occur:
  1. The expression on the right-hand side,  $x + 1$ , is computed first using the current value of  $x$ .
  2. The result becomes the new value of  $x$ .

```
x = 3          # value of x is 3
x = x+1        # x = 3+1
print(x)
```

4

```
x = 25          # value of x is 25
x = x-3        # x = 25-3
print(x)
```

22

```
x = 10          # value of x is 10
x = x*2        # x = 10*2
print(x)
```

[Skip to main content](#)

20

```
x = 15      # value of x is 15
x = x/3      # x = 15/3
print(x)
```

5.0

- Since these types of assignments are used frequently, there is a shorter version for them.

**regular**

`x = x+2`

`x = x-2`

`x = x*2`

`x = x/2`

**shorter**

`x+=2`

`x-=2`

`x*=2`

`x/=2`

```
x=3      # value of x is 3
x += 1    # x = 3+1
print(x)
```

4

```
x = 25      # value of x is 25
x -= 3      # x = 25-3
print(x)
```

22

```
x = 10      # value of x is 10
x *= 2      # x = 10*2
print(x)
```

[Skip to main content](#)

20

```
x = 15      # value of x is 15
x /= 3      # x = 15/3
print(x)
```

5.0

## Built-in Functions

Python has many useful functions available for use without importing additional modules.

- You can find the list of built-in functions in the [official Python documentation](#).
- Some of the built-in functions related to mathematics include:
  - `abs()`: returns the absolute value.
  - `max()`: returns the maximum value in a given list of numbers.
  - `min()`: returns the minimum value in a given list of numbers.
  - `sum()`: returns the sum of the values in a given list of numbers.
  - `pow()`: returns a number (base) raised to a certain power.
  - `round()`: rounds a number to a certain decimal place.

```
# absolute values of -7
print(abs(-7))
```

7

```
# maximum of 1,9,2,4 is 9
print(max(1,9,2,4))
```

9

[Skip to main content](#)

```
# minimum of 1,9,2,4 is 1  
print(min(1,9,2,4))
```

1

```
# sum of 1,9,2,4 is 16  
# numbers in square brackets or parenthesis  
print(sum([1,9,2,4]))
```

16

```
# 2 to the 3rd power  
print(pow(2, 3))
```

8

```
# rounding to the nearest thousandths  
print(round(3.4678, 3))
```

3.468

## Math Module

Modules will be imported as needed, not loaded by default.

- This minimizes memory requirements and improves performance.
- Modules are single Python files with a .py extension, which may contain functions and constants.

The Math module contains commonly used mathematical functions and constants, including trigonometric functions and the constant  $\pi$ .

- The `math` module should be imported first

[Skip to main content](#)

- `help(math)` provides more details, including explanations of functions and constants.
- You do not need to memorize these functions. Whenever you need any of them, you can import them from the module.

```
import math
```

```
# list of constants and functions in math module
print(dir(math))
```

```
['__doc__', '__file__', '__loader__', '__name__', '__package__', '__spec__', 'acos', 'a
```

```
# sin(30 radians)
math.sin(30)
```

```
-0.9880316240928618
```

```
# square root
math.sqrt(49)
```

```
7.0
```

```
# converts degrees to radians
math.radians(180)
```

```
3.141592653589793
```

```
# log of 100 in base 10
math.log10(100)
```

```
2.0
```

[Skip to main content](#)

# Combine Strings and Numbers

- There are different ways of combining strings and numbers.
- One easy way is to convert numbers to a string using the `str()` function and then concatenate strings using the `+` operator.

```
x = 5  
y = 'dollars'
```

```
# ERROR: int + str  
print(x+y)
```

```
# convert x into a string: new value is '5'  
# concatenate three strings: str(x), ' ', y  
new_str = str(x) +' '+y  
print(new_str)
```

5 dollars

## Numbers Debugging

- Each of the following short code contains one or more bugs.
- Please identify and correct these bugs.
- Provide an explanation for your answer.

### Question-1

```
x = 'five'  
y = int(x)  
print(y)
```

 Solution



[Skip to main content](#)

## Question-2

```
x = '5.07'  
y = int(x)  
print(y)
```

### Solution



x is a string representation of a decimal number. It cannot be converted to an integer.

## Question-3

```
x = 5  
print(10/(5-x))
```

### Solution



Division by zero.

## Question-4

```
print(sum(4,7,9))
```

### Solution



The numbers inside the sum() function can be enclosed in square brackets or parentheses.

## Question-5

```
print(sqrt(25))
```

[Skip to main content](#)

### Solution

`sqrt()` is not a built-in function; it needs to be imported.

## Question-6

```
print(math.sqrt(25))
```

### Solution

The math module must be imported first.

## Question-7

```
x = '5'  
y = 10  
print(x+y)
```

### Solution

Numbers and strings cannot be added or concatenated.

## Question-8

```
x = 1,234  
y = 10  
print(x+y)
```

### Solution

In Python, underscores are used instead of commas for large numbers.

[Skip to main content](#)

## Question-9

```
x = 3  
y = 10  
print(xy)
```

### Solution

In Python, multiplication is performed using the `*` operator. For example, to multiply `x` and `y`, you would write `x * y`.



## Numbers Output

- Find the output of the following code.
- Please don't run the code before giving your answer.

## Question-1

```
print(type(10/2))
```

► Show code cell output

## Question-2

```
print(type(10//2))
```

► Show code cell output

## Question-3

```
print(15%4)
```

► Show code cell output

[Skip to main content](#)

## Question-4

```
print(5+12/3)
```

► Show code cell output

## Question-5

```
print(15//4)
```

► Show code cell output

## Question-6

```
print((23//3)%4)
```

► Show code cell output

## Question-7

```
print(3**4)
```

► Show code cell output

## Question-8

```
import statistics  
average = statistics.mean([1,5,12])  
print(average)
```

► Show code cell output

## Question-9

[Skip to main content](#)

```
import numpy  
average = numpy.mean([5,10,9])  
print(average)
```

► Show code cell output

## Question-10

```
rd = round(12.345,2)  
print(rd)
```

► Show code cell output

## Question-11

```
rd = round(12.367,1)  
print(rd)
```

► Show code cell output

## Question-12

```
x = 10  
x = x+1  
print(x)
```

► Show code cell output

## Question-13

```
x = 20  
x += 1  
print(x)
```

► Show code cell output

[Skip to main content](#)

```
x = 10  
x = x+1  
x += 1  
print(x)
```

► Show code cell output

## Question-15

```
x = 10  
x -= 2  
x *= 5  
x = x+1  
print(x)
```

► Show code cell output

## Question-16

```
x = 10  
x -= 7  
print(x)  
x *= 2  
print(x)
```

► Show code cell output

## Question-17

```
x = 0  
print(x*'-+'+'$'+x*'-')  
x += 1  
print(x*'-+'+'$'+x*'-')  
x += 1  
print(x*'-+'+'$'+x*'-')  
x -= 1  
print(x*'-+'+'$'+x*'-')  
x -= 1  
print(x*'-+'+'$'+x*'-')
```

► Show code cell output

[Skip to main content](#)

## Question-18

```
x = '5.07'  
y = float(int(float(x)))  
print(y)
```

► Show code cell output

## Question-19

```
x = 2  
y = 5  
print(str(x) + ' * ' + str(y) + ' = ' + str(x*y))
```

► Show code cell output

## Numbers Code

Please solve the following questions using Python code.

### Question-1

Write a program that prompts the user for their weight in pounds.

- Convert the weight to kilograms.
- Print the converted weight.

#### Solution

```
pound = float(input('Enter your weight in pounds: '))  
  
kilogram = pound*0.46  
  
print('Your weight is', kilogram, 'kg.')
```

[Skip to main content](#)

Find the area of a circle with a radius of 5.

- Round the area to the nearest hundredth.
- $\text{Area} = \pi r^2$

### Solution

▶ Show code cell content

## Question-3

Write a program that prompts the user for a Fahrenheit temperature, converts the temperature to Kelvin, and prints out the converted temperature.

- Hint:  $K = \frac{F - 32}{1.8} + 273$

### Solution

```
fahrenheit = float(input('Enter the Fahrenheit Temperature: '))
kelvin = (fahrenheit-32)/1.8+273    # convert to kelvin
print('Kelvin Temparature is', kelvin)
```

## Question-4

Write a program that prompts the user for height (cm) and weight (kg), computes the body mass index, and prints it.

- Hint:  $BMI = \frac{\text{weight}}{\text{height}^2}$

[Skip to main content](#)

## Solution

```
height = int( input('Enter the Height: ') )
weight = int( input('Enter the Weight: ') )

bmi = weight/(height**2)

print('Body Mass Index is',bmi)
```

## Question-5

Write a program that prompts the user for a 3-digit positive number and displays the sum of its digits.

- Use only one input function.

## Solution-1

```
num = int( input('Enter a 3 digit number: ') )

n1 = num%10
n2 = int(((num-n1)/10)%10)
n3 = num//100      # or n3=int((num-n1-n2)/100)

sum_digit = n1+n2+n3

print('Sum of digits is', sum_digit)
```

## Solution-2

```
num = int( input('Enter a 3 digit number: ') )

n3 = num//100
n2 = (num-n3*100)//10
n1= num%10      # or (num-n3*100-n2*10)

sum_digit = n1+n2+n3

print('Sum of digits is', sum_digit)
```

[Skip to main content](#)

## Question-6

Write a program that prompts the user for three different integers one by one, sorts these numbers, and prints them from smallest to largest.

- Use the built-in functions `max()` and `min()`, and do not use any sorting function.

### Solution-1

```
num1 = int ( input('Enter the First Number: ') )
num2 = int ( input('Enter the Second Number: ') )
num3 = int ( input('Enter the Third Number: ') )

minimum = min(num1,num2,num3)
maximum = max(num1,num2,num3)
middle = num1+num2+num3-minimum-maximum

print(minimum,middle,maximum,sep = ',')
```

### Solution-2

```
num1 = int ( input('Enter the First Number: ') )
num2 = int ( input('Enter the Second Number: ') )
num3 = int ( input('Enter the Third Number: ') )

minimum = min(num1,num2,num3)
maximum = max(num1,num2,num3)

middle = min(max(num1,num2), max(num2,num3), max(num3,num1))

print('Smallest to largest integer: ',min_num,mid,max_num)
```

## Question-7

Write a program that prompts the user for a 2-digit positive number. Swap the digits of the given number and print it.

- Example 1: If the given number is 53, then print 35.
  - Print format: 53 — swap—> 35

[Skip to main content](#)

- Print format: 71 ---- swap---> 17

### 🔔 Solution

```
number = int(input("Enter a two digit number: "))

ones = number%10
tens = number//10

result = ones*10 + tens

print(number, '---- swap--->', result)
```



## Question-8

Write a program that prompts the user for a 3-digit positive number. Swap the digits of the given number and print it.

- Example 1: If the given number is 153, then print 351.
  - Print format: 153 ---- swap---> 351
- Example 2: If the given number is 571, then print 175.
  - Print format: 571 ---- swap---> 175

### 🔔 Solution

```
number = int(input("Enter a three digit number: "))

ones = number%10
tens = (number//10)%10
hundreds = number//100

result = ones*100 + tens*10 + hundreds

print(number, '---- swap--->', result)
```



## Question-9

[Skip to main content](#)

Write a program that prompts the user for two numbers using two input functions. Assign the entered values to variables with the names x and y.

- Find  $f(x, y) = \frac{2x^3 + 3y^2}{x^2 + y^2 + 1} \times 5$
- Round this value to the second decimal place and print the result in the following format.
  - for  $x = 1, y = 2$  display  $f(1, 2) = 11.67$

### Solution

```
x = int(input('x: '))
y = int(input('y: '))

f_x_y = (2*x**3+3*y**2)/(x**2+y**2+1)*5

print('f(', x, ', ', y, ')=' , round(f_x_y,2), sep='')
```

## Question-10

Write a program that prompts the user for 5 numbers using five different input functions.

- Find the average of these numbers without using any built-in function.
- Round it to the nearest integer and display the result.
- Example: If the given numbers are 2, 8, 9, 6, 5, then display  $\text{average}(2, 8, 9, 6, 5) = 6$ .

### Solution

```
number1 = float(input("Enter 1st number: "))
number2 = float(input("Enter 2nd number: "))
number3 = float(input("Enter 3rd number: "))
number4 = float(input("Enter 4th number: "))
number5 = float(input("Enter 5th number: "))

average = sum([number1, number2, number3, number4, number5])/5

print('average( ', number1, ', ', number2, ', ', number3, ', ', number4, ', ', number5, ') = ', round(average,0))
```

Write a program that prompts the user for 5 numbers using five different input functions.

- Find the sum of these numbers excluding the largest and smallest ones.
- Display the sum.

### Solution

```
number1 = float(input("Enter 1st number: "))
number2 = float(input("Enter 2nd number: "))
number3 = float(input("Enter 3rd number: "))
number4 = float(input("Enter 4th number: "))
number5 = float(input("Enter 5th number: "))

max_number = max(number1, number2, number3, number4, number5)
min_number = min(number1, number2, number3, number4, number5)

total = sum([number1, number2, number3, number4, number5])

print(total-(max_number+min_number))
```

## Question-12

- Choose a 3-digit random number as the dividend.
- Choose a 1-digit random number as the divisor.
- Display these numbers.
- Find the remainder and quotient when the dividend is divided by the divisor.
- After 5 seconds, show the correct remainder and quotient.

### Solution

▶ Show code cell content

## Numbers Exercises

### Question-1

[Skip to main content](#)

## Question-2

In a single line of code, compute  $|ln(2^{\sin(|-100|)})|$  and round it to the nearest hundredth.

## Question-3

Write a program that prompts the user for three numbers using three input functions. Assign the entered values to variables with the names x, y, and z.

- Find  $f(x, y, z) = \frac{5xy}{2 + x^2} + \frac{x + y + z}{y^4 + x^2y^2}$
- Round this value to the nearest hundredth and print the result in the following format.
- Sample Output:  
x: 1  
y: 2  
z: 3  
 $f(1,2,3)=3.63$

## Question-4

Write a program that prompts the user to enter their height using two input() functions for the feet and inch parts separately. Assign the entered values to variables named feet and inch.

- Convert the given height into centimeters using the following conversion formulas: 1 foot = 12 inches and 1 inch = 2.54 cm
- Sample Output:  
Enter the feet part of your height: 6  
Enter the inch part of your height: 4  
6 feet and 4 inches = 193.04 cm

## Question-5

Write a program that prompts the user for a 4-digit positive number. Swap the first two digits of the given number with the last two digits and print it.

- Example 1: If the given number is 1234, then print 3412.
  - Print format: 1234 — swap—> 3412
- Example 2: If the given number is 6789, then print 8967.
  - Print format: 6789 — swap—> 8967

## Question-6

In a coordinate plane, each point is represented by its x and y components in the form of  $(x, y)$ . The distance between two points  $P = (x_1, y_1)$  and  $Q = (x_2, y_2)$  is given by the following distance formula:

- $dist(P, Q) = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$

Write a program that prompts the user for the x and y components of a point using two *input()* functions and computes the distance between the given point and the point  $(-5, 6)$ .

- Round the distance to the nearest hundredth.

Sample Output:

Enter the x-component of the point: -2

Enter the y-component of the point: 10

The distance between (-5,6) and (-2.0,10.0) is 5.0

## Chp-4: Strings

- Learning Objectives
  - ..
  - ..

## Create Strings

Strings are ordered sequences of characters. There are several ways to create strings:

- Single quotes: `'text'`

[Skip to main content](#)

- Triple single quotes: `'''text'''`
- Triple double quotes: `""""text""""`

A space is also a character that can be in a string. Triple single and double quotes are used to create strings spanning multiple lines.

If a single quote is a character in the string, using single quotes to create the string is an error. This is because the character single quote will end the creation of the string. It behaves like the second single quote, instead of being a character in the string.

You can use double quotes to create the string to overcome the confusion. Alternatively, you can use escaping characters.

```
# Single quotes  
name = 'Mary'
```

```
# ERROR: Double single quotes cannot be used to create strings.  
name = ''Mary''  
print(name)
```

```
# Double quotes  
name = "Mary"
```

```
# Triple single quotes  
name = '''Mary'''
```

```
# Triple double quotes  
name = """"Mary""""
```

```
# ERROR  
'Mary's book.'
```

- In the code above, the second single quote ends the creation of the string.
- The `s book.'` part causes the error because it has only one single quote.
- If you use double quotes, then there will not be any problem.

[Skip to main content](#)

```
# no ERROR
sentence = "Mary's book."
```

```
# ERROR
"Mary said "I am here"."
```

- In the code above, the second double quote ends the creation of the string.
- The `I am here` part causes the error because it is not enclosed by single or double quotes.
- If you use single quotes, then there will not be any problem.

```
# no ERROR
sentence = 'Mary said "I am here'. '
```

```
# ERROR: Simple quotes cannot be used with strings spanning more than one line.
name = 'John
      Steinbeck'
print(x)
```

```
# ERROR: Double quotes cannot be used with strings spanning more than one line.
name = "John
      Steinbeck"
```

```
# no ERROR
name = '''John
      Steinbeck'''
```

```
# no ERROR
name = """John
      Steinbeck"""
```

## Escaping characters

An escape character is a backslash `\` followed by a character. It is used to write special characters in a string.

[Skip to main content](#)

- `\'` : Single quote (apostrophe)
- `\\"` : Double quote
- `\n` : New line
- `\t` : Tabulation
- `\b` : Backspace
- `\\\` : Backslash
- `\r` : Carriage return

```
# \' is the character '
print('Mary\'s book.')
```

Mary's book.

```
# \\" is the character "
print("Mary said \\\"I am here\\\".")
```

Mary said "I am here".

```
# \n is a new line
print('John\nSteinbeck')
```

John  
Steinbeck

```
# \t is a tab
# Inserts spaces up to the next tab stop, which occurs every 8th character.
print('John\tSteinbeck')
print('John'+' '*4+'Steinbeck')
```

John Steinbeck
John Steinbeck

[Skip to main content](#)

```
print('Mark\tTwain')
print('Mark'*4+'Twain')
```

Mark Twain  
Mark Twain

- `\b` (backspace) moves one character back, deleting the preceding character.

```
print('John\b Steinbeck')
```

John Steinbeck

- `\b\b` (two backspaces) moves two characters back, deleting the preceding two characters.

```
print('John\b\b Steinbeck')
```

John Steinbeck

```
# \\ is the \ (backslash) character
print('John\\ Steinbeck')
```

John\ Steinbeck

## Raw Strings

- Each character in a raw string has no special meaning; they are just characters.
- It is created using `r` in front of the string: `r'text'`
- In the example below, `\t, \n, \', \"` have no special meanings; they are just characters `\, t, n, ', "`

```
# \t means two characters, \ and t.  
# \t does not mean a tab in a raw string  
  
text = r'Good \t bye.'  
print(text)
```

Good \t bye.

```
text = r'Hello.\t my name\n is Tom. I am\' from\" England.'  
print(text)
```

Hello.\t my name\n is Tom. I am\' from\" England.

## f-strings

- It is a great way to combine constants and variable values.
- It is in the form of: `f'text {variable} text'`
- Variables are enclosed in curly brackets `{variable}` called placeholders.
- An f-string generates a new string.
- You can also perform rounding or algebraic operations within curly brackets.
- It is much easier to use f-strings than comma-separated values in a print() function.

```
name = 'Tom'  
country = 'Spain'  
age = 25  
weight = 173.6294
```

- You can compare the next two cells below to see the advantage of using f-strings.

```
# using an f-string  
print(f'My name is {name}.')
```

My name is Tom.

[Skip to main content](#)

```
# longer way  
print('My name is ', name, '.', sep='')
```

My name is Tom.

```
# using an f-string  
  
text = f'My name is {name}.' # text is a string  
print(text)
```

My name is Tom.

```
# Multiple placeholders  
print(f'My name is {name} and I am from {country}. I am {age} years old.')
```

My name is Tom and I am from Spain. I am 25 years old.

```
# algebraic operation inside curly brackets  
print(f'I will be {age+1} years old next year.')
```

I will be 26 years old next year.

```
# rounding by using the round() function  
  
print(f'My weight is {weight}.')  
print(f'My rounded weight is {round(weight,2)}.)
```

My weight is 173.6294.  
My rounded weight is 173.63.

```
# rounding by a different way  
  
print(f'My weight is {weight}.')  
print(f'My rounded weight is {weight:.2f}.') # .2 means second decimal place (hundredths)
```

[Skip to main content](#)

```
My weight is 173.6294.  
My rounded weight is 173.63.
```

## Unicode Characters

- These are symbols, accented letters, non-Latin characters, and emojis—kind of different characters.
- You can find the list of Unicode characters on the official website of [Unicodes](#).
- Each Unicode character has a code that is unique to it.
  - If the code has four characters, use
    - `\uXXXX` where XXXX is the code.
  - If a code has four characters or more, pad it with 0 from the left to make the length of the code eight and use:
    - `\Uxxxxxxxxx` where xxxxxxxx is the 0-padded code.

```
# unicode code is 1F639, you need to add 3 zeros to the left  
print('\U0001F639')
```



```
# unicode code is 1F602, you need to add 3 zeros to the left  
print('\U0001F602')
```



```
# unicode code is 2764  
print('\u2764')
```



```
# unicode code is 2764. you need to add 4 zeros to the left
```

[Skip to main content](#)

# Operations on strings

## Concatenation

- The `+` operator is used to concatenate two strings.
- String + String: combines two strings.

```
x = 'John'
y = 'Steinbeck'
```

```
# concatenation of x and y
name = x+y
print(name)
```

JohnSteinbeck

```
# add a space between x and y
# concatenation of x, space, and y

name = x+' '+y
print(name)
```

John Steinbeck

## Repetition

- The `*` operator is used to repeat a string a certain number of times.
- `String * Integer` or `Integer * String` makes copies of the string Integer many times.
- Floats cannot be used for repetitions.

[Skip to main content](#)

```
# four copies of x  
fourtoms = x*4  
print(fourtoms)
```

JohnJohnJohnJohn

```
# ERROR: float * str  
# floats can not be used for repetition  
4.3*'Hi'
```

```
# Triangle with the '$' character using repetitions.  
print('$')  
print('$'*2)  
print('$'*3)  
print('$'*4)  
print('$'*5)  
print('$'*6)  
print('$'*7)
```

\$  
\$\$  
\$\$\$  
\$\$\$\$  
\$\$\$\$\$  
\$\$\$\$\$\$  
\$\$\$\$\$\$\$

```
# Triangle with the '$' and ' ' (space) characters using repetitions."  
print(' '*6+'$')  
print(' '*5+'$'*2)  
print(' '*4+'$'*3)  
print(' '*3+'$'*4)  
print(' '*2+'$'*5)  
print(' '*1+'$'*6)  
print(' '*0+'$'*7)    # you do not have to include the space part because it adds no sp
```

[Skip to main content](#)

```
$  
$$  
$$$  
$$$$  
$$$$$  
$$$$$  
$$$$$
```

## Length Function

- The built-in `len()` function returns the number of characters in a string.

```
# there are five characters in hello  
print(len('hello'))
```

5

```
# there are six characters in 'hel lo'  
# space is a character  
print(len('hel lo'))
```

6

```
# there are six characters in 'hel\nlo'  
# \n is a new line character (single character)  
print(len('hel\nlo'))
```

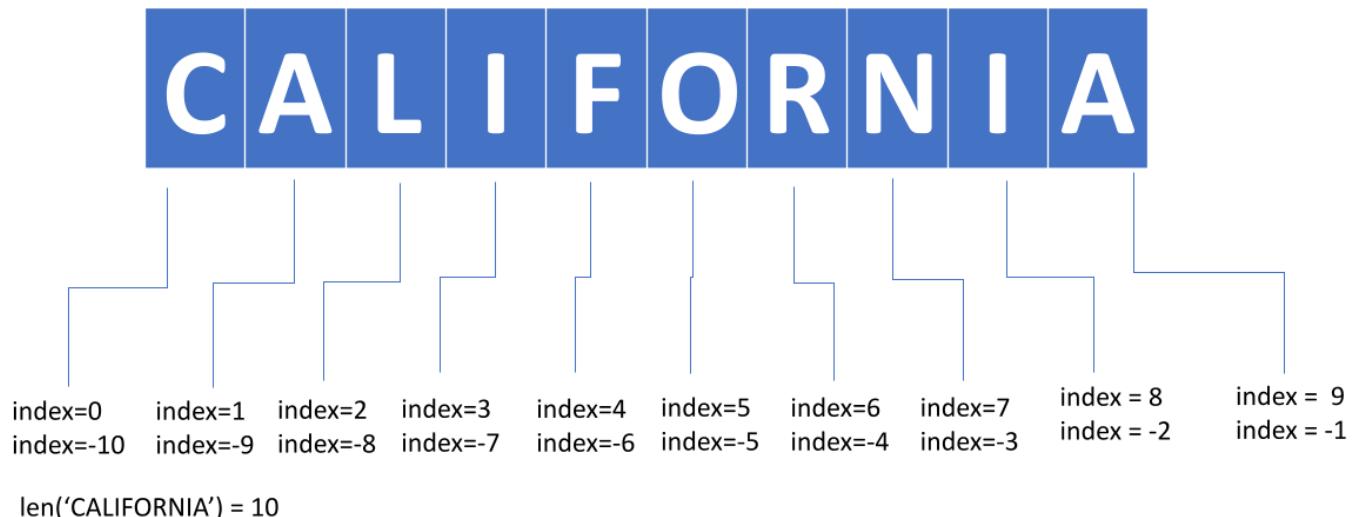
6

## String Indexing

- Indexing is used to access individual characters or sets of characters.
- Indexing starts with zero

[Skip to main content](#)

- The index of the second character is 1, and so on.
- The index is written in square brackets: string[index].
- Negative numbers can also be used for indexing.
- The index of the last character is -1.
- The index of the second character from the end is -2, and so on.



- index of **C** is 0 or -10
- index of **first A** is 1 or -9
- index of **second A** is 9 or -1
- **Warning:** There is a character with index -10 (the second 'A'), but there is no character with index positive 10 because indexing starts with 0, and it does not reach 10, which is the length of the string.
  - For any string, there is no character with an index equal to the length of the string.

```
state = 'CALIFORNIA'
```

```
# access the character at index 0 by using square brackets."
```

[Skip to main content](#)

C

```
# Access the character at index 6 by using square brackets."  
print(state[6])
```

R

```
# Access the character at index -1 by using square brackets."  
print(state[-1])
```

A

```
# Access the character at index -3 by using square brackets."  
print(state[-3])
```

N

- There is an error in the following code because there is no character with index 10.

```
# ERROR: out of range  
state[10]
```

- The length of state is 10, and there is an error in the following code. This applies to all strings

```
# ERROR: out of range  
state[len(state)]
```

```
# There is no error in the following code because len(state) - 1 = 9 is the index of th  
print(state[len(state)-1])
```

A

# String Slices

- You can access more than one character of a string by using index numbers.
- It is in the form of `string[start: end]` with inclusive start and exclusive end.
- Use `:` (colon) inside square brackets between the start and end indexes.
- It consists of characters starting with index start up to the character with index end-1.
- The character with index end is not included.
- For example, `string[2:5]` returns characters with indexes 2, 3, 4 (5 is not included).
- For example, `string[-4:-1]` returns characters with indexes -4, -3, -2 (-1 is not included).
- It returns a substring.

index	0	1	2	3	4	5	6	7	8	9		
	C	A	L	I	F	O	R	N	I	A		
	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1		negative index

```
state = 'CALIFORNIA'
```

```
print(state[2:5]) # index=2,3,4
```

LIF

[Skip to main content](#)

```
print(state[-4:-1]) # index=-4,-3,-2
```

RNI

- `string[:end]`: the default value of start is 0, which means it starts from the very beginning.
  - For example, `string[:5]` returns characters with indexes 0, 1, 2, 3, 4 (5 is not included).
- `string[start:]`: the default value of end is the length, which means it goes all the way to the end.
  - For example, `string[2:]` returns characters with indexes 2, 3, 4, 5, 6, 7, 8, 9 (all characters starting from index 2).
- `string[:]`: starting from the very beginning and going all the way to the end, representing the whole string.

```
print(state[2:]) # index = 2,3,4,5,6,7,8,9
```

LIFORNIA

```
print(state[:5]) # index = 0,1,2,3,4
```

CALIF

```
print(state[:]) # index = all of them = 0,1,2,...,9
```

CALIFORNIA

- Slicing can also be done by taking steps in the form of: `string[start: end: step]`.
- `string[start: end: step]` means starting with the character at index = start up to the character at index = length - 1, as before, but not necessarily including all characters between them.
- The first index is start, the second index is start + step, and the third index is start + 2 \* step.
- It continues in this way, but the largest index can be at most length - 1.

[Skip to main content](#)

- The default value of step is 1.

```
print(state)
```

CALIFORNIA

```
print(state[2:7:2]) # index = 2,2+2=4, 4+2=6
```

LFR

```
print(state[1:8:3]) # index = 1,1+3=4, 4+3=7
```

AFN

```
print(state[7:2:-2]) # index = 7,7+(-2)=5,5+(-2)=3
```

NOI

```
print(state[-8:-2:3]) # index = -8, -8+3=-5
```

LO

```
# for negative step default value of start is 9 (-1)
# for negative step default value of end    is 0 (-10)
print(state[::-1]) # index = 9,8,...,0
```

AINROFILAC

```
print(state[-3::-1]) # index = -3,-4,...,-10
```

[Skip to main content](#)

NR0FILAC

```
print(state[::-4:-1]) # index = 9,8,7 or -1,-2,-3
```

AIN

## String module

- It contains constants and functions to process strings, as well as some constants.
- Use `help(string)` for more explanations.

```
# constants and functions  
  
import string  
print(dir(string))
```

```
['Formatter', 'Template', '_ChainMap', '__all__', '__builtins__', '__cached__', '__doc__'
```

```
# lowercase letters  
print(string.ascii_lowercase)
```

abcdefghijklmnopqrstuvwxyz

```
# lowercase and uppercase letters  
print(string.ascii_letters)
```

abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ

```
# digits  
print(string.digits)
```

[Skip to main content](#)

```
0123456789
```

```
# punctuations  
print(string.punctuation)
```

```
!"#$%&' ()*+, -./:; <=>?@[\]^_`{|}~
```

## Immutable

- Strings are immutable, which means they cannot be modified.
- For example, if you try to change the first character of 'CALIFORNIA', you will get an error message.

```
# ERROR: try to change the first character, which has an index of 0.  
state = 'CALIFORNIA'  
state[0] = 'R'
```

- You can use the state variable to produce a new string without changing the original one.
- In the following code:
  - The value of the variable *new\_state* is the concatenation of the string 'R' and a slice of *state* starting from the character at index 1 and continuing to the end.

```
state = 'CALIFORNIA'  
new_state = 'R' + state[1:]  
  
print(new_state)
```

```
RALIFORNIA
```

- In the following code, a new value is assigned to the variable *state*.

```
# 'CALIFORNIA' is not modified.  
  
state = 'CALIFORNIA'  
state = 'R' + state[1:]
```

[Skip to main content](#)

## in and not in

- These operators are used to check if a character or slice is present in a string.
- They return a boolean value: True or False.
- Python is case-sensitive.

```
print( 'a' in 'FLORIDA' ) # 'a' is not in FLORIDA
```

False

```
print( 'a' not in 'FLORIDA' ) # 'a' is not in FLORIDA
```

True

```
print( 'A' in 'FLORIDA' ) # 'A' is in FLORIDA
```

True

```
print( 'A' not in 'FLORIDA' ) # 'a' is not in FLORIDA
```

False

## String Methods

- String methods do not modify the original string because strings are immutable.
- String methods return a new value.

[Skip to main content](#)

- If you run `dir(str)`, you will see that there are many methods because there can be so many things that can be done with strings.
- We will cover some of them here, but you can check `help(str)` for more details.

## capitalize()

- Produces a duplicate of the string where only the initial character is in uppercase, while all other characters are converted to lowercase.

```
text = 't0m aNd jerRy.'  
print(text.capitalize())      # 't' is capitalized, a new string is produced  
print(text)                  # no change on text (immutable)
```

Tom and jerry.  
t0m aNd jerRy.

## upper()

- Produces a duplicate of the string where all characters are converted to uppercase.

```
text = 't0m aNd jerRy.'  
print(text.upper())          # all characters are in uppercase  
print(text)                  # no change on text (immutable)
```

TOM AND JERRY.  
t0m aNd jerRy.

## lower()

- Produces a duplicate of the string where all characters are converted to lowercase.

```
text = 't0m aNd jerRy.'  
print(text.lower())          # all characters are in lowercase  
print(text)                  # no change on text (immutable)
```

[Skip to main content](#)

```
tom and jerry.  
t0m aNd jerRy.
```

## title()

- Produces a duplicate of the string where all words are capitalized.

```
text = 't0m aNd jerRy.'  
print(text.title())          # all words are capitalized  
print(text)                 # no change on text (immutable)
```

```
Tom And Jerry.  
t0m aNd jerRy.
```

## find()

- It provides the earliest occurrence of a given substring within a string.
- It returns the lowest index.
- If the substring is not present, it returns -1.
- Additionally, you have the option to begin the search from a specific character to find the index of the given substring.
  - `find('a', N)`: find index of first 'a' starting from index=N
  - default value of N is 0

```
state = 'CALIFORNIA'  
print(state.find('L')) # index of 'L' is 2
```

2

```
# -1 means 'W' does not exist, -1 does not represent an index  
print(state.find('W'))
```

-1

```
print(state.find('A')) # index of first 'A'
```

1

```
# The index of the first occurrence of 'A' starting from the character at index 3.  
print(state.find('A', 3))
```

9

```
# The string 'FOR' begins from the character at index 4.  
print(state.find('FOR'))
```

4

```
print(state.find('WE')) # 'WE' does not exist in CALIFORNIA
```

-1

## rfind()

- It returns the maximum index in a string where the substring is located.

```
print(state.find('A')) # index of first 'A'  
print(state.rfind('A')) # index of last 'A'
```

1  
9

## strip(), rstrip(), lstrip()

- `strip()`: Removes white spaces from the beginning and end of a string.
- `rstrip()`: Removes white spaces from the end of a string.
- `lstrip()`: Removes white spaces from the beginning of a string.

```
country = ' FLORIDA '
print(country)
print('---'+country.strip()+'---')      # white spaces on the left and right are removed
print('---'+country.rstrip()+'---')      # white spaces on the right are removed
print('---'+country.lstrip()+'---')      # white spaces on the left are removed
```

```
FLORIDA
---FLORIDA---
--- FLORIDA---
---FLORIDA ---
```

## startswith()

- It returns True if the string starts with the specified prefix; otherwise, it returns False.

```
print(state.startswith('H')) # 'CALIFORNIA' does not startswith 'H'
```

```
False
```

```
print(state.startswith('C')) # 'CALIFORNIA' does startswith 'C'
```

```
True
```

## count()

- Returns the number of non-overlapping occurrences of a substring within a string

```
print('CALIFORNIA'.count('A')) # count of 'A' in 'CALIFORNIA'
```

[Skip to main content](#)

2

```
print(state.count('I')) # number of 'I's in CALIFORNIA'
```

2

```
print(state.count('C')) # number of 'C's in 'CALIFORNIA'
```

1

```
print(state.count('W')) # number of 'C's in 'CALIFORNIA'
```

0

## isdigit()

- Returns True if the string consists of digits, False otherwise.

```
print('hello'.isdigit()) # not all characters are digits
```

False

```
print('123456'.isdigit()) # all characters are digits
```

True

```
print('h1234'.isdigit()) # not all characters are digits
```

[Skip to main content](#)

False

## isalpha()

- Returns True if the string consists of alphabetic characters, False otherwise.

```
print('hello'.isalpha()) # all characters are alphabetic
```

True

```
print('123456'.isalpha()) # not all characters are alphabetic
```

False

```
print('h1234'.isalpha()) # not all characters are alphabetic
```

False

## replace()

- Returns a duplicate with all occurrences of the old substring replaced by the new one.
- It is in the form of `replace(old, new)`

```
print(state)
print(state.replace('A', 'W')) # replace 'A' by 'W'
```

CALIFORNIA  
CWLIFORNIW

```
print(state)
print(state.replace('T', 'W')) # no 'T' to replace by 'W'
```

CALIFORNIA  
CALIFORNIA

```
print(state)
print(state.replace('LI', '**')) # replace 'LI' by '**'
```

CALIFORNIA  
CA\*\*\*FORNIA

## swapcase()

- Transform uppercase characters to lowercase and lowercase characters to uppercase.

```
name = 'aRThUr'
print(name)
print(name.swapcase()) # 'a' becomes 'A', 'R' becomes 'r', and so on
```

aRThUr  
ArtHuR

## join()

- Concatenate a list of strings.
- Insert the string, whose method is called, between each given string.
- Return the result as a new string.
- Example: `'--'.join(['ab', 'pq', 'rs'])` returns `'ab--pq--rs'`

```
print('--'.join(['ab', 'pq', 'rs']))
```

[Skip to main content](#)

# Parsing Strings

- By using string methods, you can analyze a string and extract meaningful information about the string.
- You can also perform specific operations based on the structure and content of the string.
- Example:
  - From the given message below, extract the company name and capitalize it.
  - The company name is between the characters `@` and `.`
  - We can use the `find()` method to find the indexes of these two characters.
  - There are multiple `.` characters, so we need to find the first one that comes after `@`.

```
message = 'Hello. My name is Tom. I live in California. My email address is tom@tesla.c'
```

```
index_at = message.find('@')                      # index of @  
index_period = message.find('.', index_at)         # index of first . after @
```

- To grab the company name, we need to use slicing.
- Slicing must start from `index_at + 1` because if you start from `index_at`, the slice will include `@`.
- Slicing must end at `index_period` because the end index is not included.

```
print(message[index_at+1:index_period].capitalize())
```

Tesla

# Strings Debugging

- Each of the following short code contains one or more bugs.
- Please identify and correct these bugs.
- Provide an explanation for your answer.

## Question-1

[Skip to main content](#)

```
3.0*'HI'
```

### Solution

Only integers can be used for repetition; 3.0 is a float.

## Question-2

```
3+'HI'
```

### Solution

Concatenation can be done with two strings, but 3 is not a string. You can convert 3 to a string, and after that, you can perform concatenation: `str(3) + 'HI'.`

## Question-3

```
name = 'michael"
```

### Solution

Both of the quotes on the left and right must be the same. Either `'michael'` or `"michael"` will solve the problem.

## Question-4

```
print('he's coming')
```

[Skip to main content](#)

### Solution

The single quote of the string causes the problem since the string is also created by using single quotes.

- To fix the problem you can either use `\` to make the single quote of the string a character: `'he\ 's coming'`
- Instead of single quotes, use double ones: `"he\ 's coming"`

## Question-5

```
name = 'Michael  
Jordan'  
print(name)
```

### Solution

Triple single or double quotes must be used for strings that have multiple lines.

## Question-6

```
name = 'Brian'  
print(name[5])
```

### Solution

The index is out of range because the largest positive index is 4, as indexing starts from 0.

# Strings Output

- Find the output of the following code.
- Please don't run the code before giving your answer.

[Skip to main content](#)

## Question-1

```
print('Mark'+'Twain')
```

► Show code cell output

## Question-2

```
print('Mark '+'Twain')
```

► Show code cell output

## Question-3

```
print('Mark\tTwain')
```

► Show code cell output

## Question-4

```
print('Mark\nTwain')
```

► Show code cell output

## Question-5

```
print('Mark\bTwain')
```

► Show code cell output

## Question-6

```
print('Mark\b\bTwain')
```

[Skip to main content](#)

## Question-7

```
print('Mark\rTwain')
```

► Show code cell output

## Question-8

```
print('Mary\'s book.')
```

► Show code cell output

## Question-9

```
print(r'Mary\'s \n book.')
```

► Show code cell output

## Question-10

```
text = 'Hello World.'  
len(text)
```

► Show code cell output

## Question-11

```
x = 'abcdefg'  
  
print(x[:])  
print(x[:-3])  
print(x[-4:])  
print(x[2:4])  
print(x[-4:-2])
```

► Show code cell output

[Skip to main content](#)

## Question-12

```
name = 'Mary'  
print(f'My name is {name}')
```

► Show code cell output

## Question-13

```
name = 'George'  
salary = 2000  
  
print(f'My name is {name}. I earn ${salary} per month.')
```

► Show code cell output

## Question-14

```
x = 'california'  
  
print(x.find('f'))  
print(x.find('A'))  
print(x.find('a', 4))
```

► Show code cell output

## Question-15

```
state = 'utah'  
print(3*'utah')
```

► Show code cell output

## Question-16

[Skip to main content](#)

```
name = 'MaRk tWAin'

print(name.upper())
print(name.lower())
print(name.capitalize())
print(name.title())
```

► Show code cell output

## Question-17

```
country = 'Morocco'

print(country.find('r'))
print(country.find('o'))
print(country.find('c'))
print(country.find('o',4))
print(country.find('t'))
```

► Show code cell output

## Question-18

```
name = 'Liz'
print(name.strip())
```

► Show code cell output

## Question-19

```
name = 'Liz '
print(name.lstrip()+'T')
```

► Show code cell output

## Question-20

[Skip to main content](#)

```
name = 'ryan'  
name.startswith('R')
```

► Show code cell output

## Question-21

```
word = 'ab,cde=abc,de'  
  
index1 = word.find('f')  
index2 = word.find('=')  
index3 = word.find(',', index2)  
  
print(word[index2:index3+index1])
```

► Show code cell output

## Question-22

```
state = 'Florida'  
  
x = state[1:4:2].upper().lower()  
y = state[-1:-5:-1].capitalize().swapcase()  
  
print(y+'_'+x)  
print(x+y)
```

► Show code cell output

# Strings Code

- Please solve the following questions using Python code.

## Question-1

Create a variable named `state` and assign the value `CALifoRNia` to it.

- Print the `state` in lowercase

[Skip to main content](#)

- Find the index of `R` in `state`
- Use slicing and positive index numbers to print the `ifo` part of the `state`
- Use slicing and negative index numbers to print the `ifo` part of the `state`

## Solution

▶ Show code cell content

## Question-2

Write a program that prompts the user to enter a word with 6 letters.

- Display the letters of the word in reverse order.

### Solution-1

```
word = input('Enter a word with 6 letters:')

word_reverse = word[5] + word[4] + word[3] + word[2] + word[1] + word[0]

print('Reverse Order:', word_reverse)
```

### Solution-2

```
word = input('Enter a word with 6 letters:')

word_reverse = word[-1] + word[-2] + word[-3] + word[-4] + word[-5] + word[-6]

print('Reverse Order:', word_reverse)
```

[Skip to main content](#)

### Solution-3

```
word = input('Enter a word with 6 letters:')

word_reverse = word[::-1]

print('Reverse Order:', word_reverse)
```

### Solution-4

```
word = input('Enter a word with 6 letters:')

print(f'{word[5]}{word[4]}{word[3]}{word[2]}{word[1]}{word[0]}')
```

## Question-3

Write a program that prompts the user to enter their first and last name with only one space between them. For example: Mark Twain

- Use only one input function.
- Display only the first name of the user.
- Display only the last name of the user in uppercase letters.

### Solution

```
name = input('Enter your first name and last name: ')

index_space = name.find(' ')
first_name = name[:index_space]
last_name = name[index_space+1:]

print('First Name:', first_name)
print('Last Name:', last_name.upper())
```

## Question 4

[Skip to main content](#)

Write a program that prompts the user to enter an email address in the form of user\_name@company\_name.com.

For example, for the email address [mtwain@oxfordpress.com](mailto:mtwain@oxfordpress.com), the user name is mtwain, and the company name is oxfordpress.

- Display only the user name of the user.
- Display only the company name of the user.

## Solution

### Solution-1

```
email = input('Enter your email address: ')  
  
index1 = email.find('@')  
index2 = email.find('.')  
  
user_name = email[0:index1]  
company_name = email[index1+1:index2]  
  
print('User Name:', user_name)  
print('Company Name:', company_name)
```

### Solution-2

```
email = input('Enter your email address in the form of user_name@company_name.c  
name = email.split('@')[0]  
company = email.split('@')[1].split('.')[0]  
  
print('Name:', name)  
print('Company:', company)
```

## Question-5

Write a program that prompts the user to enter an 8-digit ID number, last name, and company name, using three input functions.

[Skip to main content](#)

- lastname last four digits of the ID number @ company name.com
- Use only lowercase letters.
- For example, the email address is twain5678@oxfordpress.com for:
  - ID number: 12345678
  - Last name: Twain
  - Company: oxForD preSS
- Print the email address.

### Solution

```
id_number = input('ID number: ')
last_name = input('Last Name: ').lower()

company_name = input('Company Name: ').lower()

print(f'Email address: {last_name}{id_number[-4:]}@{company_name}.com')
```

## Question-6

- Extract the ID number of the YouTube video in the following text.
- ID number = inN8seMm7UI and it is between = and ).

```
text = '''Imyep jgsqewt okbxsq seunh many rkx vmysz ndpoz may vxabckewro topfd tqkj uew
'''
```

### Solution

### Solution

```
index_equal = text.find('=')
index_right_par = text.find(')', index_equal)

id = text[index_equal+1:index_right_par]

print(f'ID number: {id}')
```

[Skip to main content](#)

## Question-7

Write a program that asks the user to enter a word of any odd length.

- Display the first, middle, and last characters together as a single string.
- The first and last characters should be in lowercase.
- The middle character should be in uppercase.

### Solution

#### Solution

```
word = input('Enter a word: ')  
  
print(word[0].lower() + word[len(word)//2].upper() + word[-1].lower())
```



## Question-8

Display the following large X using the repetition of strings.

```
*          *  
*          *  
*          *  
*      *  
*  *  
*  
*  *  
*      *  
*          *  
*          *
```

[Skip to main content](#)

▶ Show code cell content

## Question-9

Display the following triangle by using the repetition of strings.

```
*  
***  
*****  
*****  
*****  
*****  
*****  
*****  
*****  
*****
```

## Solution

▶ Show code cell content

## Question-10

Display the following parallelogram by using the repetition of strings

```
*****  
*****  
*****  
*****  
*****  
*****  
*****  
*****  
*****
```

[Skip to main content](#)

## Solution

▶ Show code cell content

## Question-11

Display the following rectangle by using the repetition of strings

```
* * * * * * * * * *  
* * * * * * * * * *  
* * * * * * * * * *  
* * * * * * * * * *  
* * * * * * * * * *
```

## Solution

▶ Show code cell content

# Chp-5: Conditionals

- Learning Objectives
  - ..
  - ..

## Motivation

In our daily lives, our actions often depend on various conditions. For instance:

1. If the temperature is above 60, I will go hiking; otherwise, I will stay at home.
2. If I work less than 40 hours per week, my hourly rate is 25 dollars. For each additional hour, I earn 40 dollars.

[Skip to main content](#)

- If I have more than 5,000 dollars, I will go on a cruise.
- If I have between 3,000 and 5,000 dollars, I will go to Florida.
- If I have less than 3,000 dollars, I will spend time in my city.

4. In my math course, the letter grade is determined by a grading scale. For instance, getting 76 results in a C+.

The central question in this chapter is how to write a program that performs different tasks under various conditions. This allows us to answer questions such as whether to go outside or stay at home based on the temperature, how much to earn depending on weekly working hours, where to travel, or what grade a student will receive according to the grading scale.

## Conditionals

We have seen the following data types so far:

- `int`: Integers
- `float`: Floats
- `str`: Strings
- `bool`: Boolean

Since conditionals are the main subject of this chapter, booleans will be the commonly used data type. We have already encountered boolean values when working with:

- `in` and `not in` operators for strings return boolean values.
- `startswith()`, `isdigit()`, and `isalpha()` string methods also return boolean values.

## Booleans

- The values of booleans are limited to two: `True` and `False`.
- They are keywords in Python and cannot be used as variable names.
- boolean `True` (`False`) and string `'True'` (`'False'`) are different.

```
x = True  
y = False  
z = 'True'
```

[Skip to main content](#)

```
print(type(x)) # x is a boolean
```

```
<class 'bool'>
```

```
print(type(y)) # y is a boolean
```

```
<class 'bool'>
```

```
print(type(z)) # z is a string
```

```
<class 'str'>
```

## Conversion

### bool <-> int

- bool → int
  - int(True) = 1
  - int(False) = 0
- int → bool
  - bool( any integer other than 0 ) = True
  - bool(0) = False

```
print(int(True)) # int value of True is 1
```

```
1
```

```
print(int(False)) # int value of False is 0
```

[Skip to main content](#)

0

```
print(bool(6))      # bool value of 6 is True
```

True

```
print(bool(-101))    # bool value of -101 is True
```

True

```
print(bool(23.567))  # bool value of 23.567 is True
```

True

```
print(bool(0))      # bool value of 0 is False
```

False

- You can perform algebraic operations with boolean values `True` and `False`.
- In such operations, `True` takes on the integer value `1`, and `False` takes on the integer value `0`.

```
print(True+1)    # True+1=1+1=2
```

2

```
print(False*5)   # False*5=0*5=0
```

0

[Skip to main content](#)

```
print(True+4.6+True+False+10+True+False) # 1+4.6+1+0+10+1+0
```

17.6

## bool <-> float

- bool → float
  - float(True) = 1.0
  - float(False) = 0.0
- float → bool
  - bool( any float other than 0.0 ) = True
  - bool(0.0) = False

```
print(float(True)) # float value of True is 1.0
```

1.0

```
print(float(False)) # float value of False is 0.0
```

0.0

```
print(bool(23.567)) # bool value of 23.567 is True
```

True

```
print(bool(0.0)) # bool value of 0.0 is False
```

False

[Skip to main content](#)

## bool <-> str

- bool → str
  - str(True) = 'True'
  - str(False) = 'False'
- str → bool
  - bool( any string other than empty string ) = True
  - bool( empty string ) = False
    - empty string = 

```
print(str(True))      # str value of True is 'True'
```

True

```
print(str(False))      # str value of False is 'False'
```

False

```
print(bool('HELLO'))  # bool value of 'HELLO' is True
```

True

```
print(bool(''))        # bool value of '' = empty string is False
```

False

## Conditions (Boolean Expression)

An expression that evaluates to either True or False.

[Skip to main content](#)

# Comparison operators

Since `=` is used for assignment, `==` is used for comparison.

Operator	Meaning
<code>==</code>	equal (only value not type)
<code>!=</code>	not equal (only value not type)
<code>&lt;</code>	less than
<code>&lt;=</code>	less than or equal to
<code>&gt;</code>	greater than
<code>&gt;=</code>	greater than or equal to
<code>is</code>	equal (value and type)
<code>is not</code>	not equal to (value and type)

- Comparison of strings
  - Inequality operators follow the dictionary order.
  - Uppercase letters are considered before lowercase letters.
  - Digits are considered before uppercase letters.
  - The order is as follows: Digits < Uppercase letters < Lowercase letters.
- The `is` keyword checks for identity.
  - Objects compared with `is` have the same `id()` numbers, indicating they are stored in the same location in memory.
  - Using `is` means the objects point to the exact same object.
- `==` checks for equality
  - The values are same or not.
  - 4 and 4.0 as mathematical values same even though their type is different
- The `==` operator checks for equality.
  - It compares whether the values are the same or not.

- When you use `is` with strings, numbers, or booleans, a syntax error occurs.
  - The error suggests using `==` or `!=` for these types.

```
print(5==5) # 5 is equal to 5 is True
```

True

```
print(5==7) # 5 is equal to 7 is False
```

False

```
print(3==3.0) # The mathematical value of 3 and 3.0 are equal is True
```

True

```
print(5!=5) # 5 is not equal to 5 is False
```

False

```
print(5!=7) # 5 is not equal to 7 is True
```

True

```
print(5<7) # 5 is less than 7 is True
```

True

```
print(10>15) # 10 is greater than 15 is False
```

[Skip to main content](#)

False

```
print('k'=='t') # 'k' is equal to 't' is False
```

False

```
print('k'=='k') # 'k' is equal to 'k' is True
```

True

```
print('K'=='k') # 'k' is equal to 'K' is False (case sensitive).
```

False

```
print('k' < 't') # 'k' comes before 't' in dictionary order is True
```

True

```
print('z' < 't') # 'z' comes before 't' in dictionary order is False
```

False

```
print('money' < 'table') # 'money' comes before 'table' in dictionary order is False
```

True

```
print('Z' < 'a') # capital letters come before lower case letters in dictionary order
```

[Skip to main content](#)

True

```
print('3' < 'A') # digits come before letters in dictionary order
```

True

```
# SyntaxWarning: Use == instead
print( 'k' is 'K')
```

```
# SyntaxWarning: Use == instead
print( 2 is 3)
```

## if statement

It is used to execute a block of code depending on a condition, which is a boolean expression. So, it returns either True or False.

- If the condition is True, a block of code will be executed.
- If the condition is False, the block of code will not be executed and will be skipped.
- The structure of an if statement is as follows:

```
if condition:
```



- In the structure above:
  - `if` is a keyword.
  - `condition` is a boolean expression which is boolean True or False.
  - `:` comes right after the condition and it means this line will be followed by another code
  - `BLOCK CODE` is an group of code with same indentation level that will be executed if condition is True.

[Skip to main content](#)

- if is a keyword.
- condition is a boolean expression, which is either True or False.
- `:` comes right after the condition and means this line will be followed by another code.
- `BLOCK CODE` is a group of code with the same indentation level that will be executed if the condition is True.

The cases are as follows:

1. condition is True:

```
if True:  
    [ ]  
BLOCK CODE    BLOCK CODE will be executed.  
    [ ]
```

2. condition is False:

```
if False:  
    [ ]  
BLOCK CODE    BLOCK CODE will be skipped.  
    [ ]
```

In the code below:

- `condition` is True because  $75 > 65$ , so the block code will be executed.
- `BLOCK CODE` consists of two lines of code, and they will be executed.

```
grade = 75  
  
if grade > 65:  
    print('You passed.')  
    print('Congrats!')
```

You passed.  
Congrats!

- In the code below:

- `condition` is False because  $55 > 65$  is False, so the block code will not be executed.

[Skip to main content](#)

- There is no output for this code.

```
grade = 55  
  
if grade > 65:  
    print('You passed.')  
    print('Congrats!')
```

- In the code below:

- **condition** is False because  $55 > 65$  is False, so the block code will not be executed.
- **BLOCK CODE** consists of two lines of code and they will be skipped.
- The last print statement is not part of the **BLOCK CODE**. After skipping the **BLOCK CODE**, this last print statement will be executed.

```
grade = 55  
  
if grade > 65:  
    print('You passed.')  
    print('Congrats!')  
print('Bye')
```

Bye

- As a condition, numbers, strings, and boolean values can be directly used.
- Python will automatically convert them into boolean values.

```
# condition is always True  
# Hello will always be printed.  
  
if True:  
    print('Hello')
```

Hello

```
# condition is always False: print statement will be skipped.  
# no output  
  
if False:  
    ...
```

[Skip to main content](#)

```
# condition is always True because bool(5)=True  
# Hello will always be printed  
  
if 5:  
    print('Hello')
```

Hello

```
# condition is always False because bool(0)=False  
# no output  
  
if 0:  
    print('Hello')
```

```
# condition is always True because bool('NY')=True  
# Hello will always be printed.  
  
if 'NY':  
    print('Hello')
```

Hello

```
# condition is always False because bool('')=False  
# no output  
  
if '':  
    print('Hello')
```

- If the block code of an if statement consists of only one line, it can be written right after the colon (:), keeping the entire if statement in a single line.

```
grade = 55  
if grade > 35: print('You passed.')
```

You passed.

[Skip to main content](#)

This is for two-case situations.

- If the condition is True, the block code of the if statement will be executed as before.
- If the condition is False, the block code of the else statement will be executed.
- The `else` keyword is used for the second part.
- The else part does not have a condition part.
- **IMPORTANT:** The indentation level of if and else must be the same.

`if condition:`

`BLOCK CODE of IF`

`else:`

`BLOCK CODE of ELSE`

- The cases are as follows:

1. condition is True:

`if True:`

`BLOCK CODE of IF`

BLOCK CODE of IF STATEMENT will be executed.

`else:`

`BLOCK CODE of ELSE`

BLOCK CODE of ELSE STATEMENT will be skipped.

2. condition = False:

`if False:`

`BLOCK CODE of IF`

BLOCK CODE of IF STATEMENT will be skipped.

`else:`

[Skip to main content](#)

### BLOCK CODE of ELSE

BLOCK CODE of ELSE STATEMENT will be executed.

- In the code below:

- `condition` is True because  $75 > 65$  is True, so the block code of the `if` statement will be executed.
  - The block code of the `else` statement will be skipped.
  - The output comes from the print statements of the `if` part.

```
grade = 75

if grade > 65:
    print('You passed.')
    print('Congrats!')
else:
    print('You failed.')
    print('I am sorry.')
```

You passed.  
Congrats!

- In the code below:

- `condition` is False because  $55 > 65$  is False, so the block code of the `else` statement will be executed.
  - The block code of the `if` statement will be skipped.
  - The output comes from the print statements of the `else` part.

```
grade = 55

if grade > 65:
    print('You passed.')
    print('Congrats!')
else:
    print('You failed.')
    print('I am sorry.')
```

You failed.  
I am sorry.

- If the block code of an if or else statement consists of only one line, it can be written right after the colon (:), keeping the entire if or else statement in a single line.

```
grade = 55  
if grade > 35: print('You passed.')  
else: print('You failed.')
```

You passed.

## if-elif-else statement

This is for three-case situations.

- If there are more cases, then more `elif` parts can be added.
- If the condition of the `if` statement is True, the block code of the `if` statement will be executed.
- If the condition of the `if` statement is False and the condition of the `elif` statement is True, the block code of the `elif` statement will be executed.
- If the conditions of the `if` and `elif` statements are False, then the block code of the `else` statement will be executed.
- The `elif` keyword is used for the second part.
- `elif` has a condition part.
- **IMPORTANT:** The indentation level of `if`, `elif`, and `else` must be the same.
- The structure of an if-elif-else statement is as follows:



[Skip to main content](#)

BLOCK CODE of ELSE

- The cases are as follows:

- condition of `if` statement is True:

`if True:`

BLOCK CODE of IF

BLOCK CODE of IF STATEMENT will be executed.

`elif condition of ELIF:`

BLOCK CODE of ELIF

BLOCK CODE of ELIF STATEMENT will be skipped.

`else:`

BLOCK CODE of ELSE

BLOCK CODE of ELSE STATEMENT will be skipped.

- condition of `if` statement is False and condition of `elif` statement is True:

`if False:`

BLOCK CODE of IF

BLOCK CODE of IF STATEMENT will be skipped.

`elif True:`

BLOCK CODE of ELIF

BLOCK CODE of ELIF STATEMENT will be executed.

`else:`

BLOCK CODE of ELSE

BLOCK CODE of ELSE STATEMENT will be skipped.

- conditions of `if` and `elif` statements are both False:

`if False:`

[Skip to main content](#)

BLOCK CODE of IF

BLOCK CODE of IF STATEMENT will be skipped.

elif False:

BLOCK CODE of ELIF

BLOCK CODE of ELIF STATEMENT will be skipped.

else:

BLOCK CODE of ELSE

BLOCK CODE of ELSE STATEMENT will be executed.

- In the code below:

- condition of the if statement is True because  $75 > 65$  is True, so the block code of the if statement will be executed.
- The block code of the elif and else statements will be skipped.
- The output comes from the print statements of the if part.

```
grade = 75

if grade > 65:
    print('You passed.')
    print('Congrats!')
elif grade > 55:
    print('You could not pass!')
    print('You can take the test one more time.')
else:
    print('You failed.')
    print('I am sorry.')
```

You passed.  
Congrats!

- In the code below:

- condition of the if statement is False because  $60 > 65$  is False
- condition of the elif statement is True because  $60 > 55$  is True
- So the block code of the elif statement will be executed.
- The block code of the if and else statements will be skipped.

```
grade = 60

if grade > 65:
    print('You passed.')
    print('Congrats!')
elif grade > 55:
    print('You could not pass!')
    print('You can take the test one more time.')
else:
    print('You failed.')
    print('I am sorry.')
```

You could not pass!  
You can take the test one more time.

- In the code below:
  - **condition** of the if statement is False because  $40 > 65$  is False
  - **condition** of the elif statement is False because  $40 > 55$  is False
  - So the block code of the else statement will be executed.
  - The block code of the if and elif statements will be skipped.
  - The output comes from the print statements of the else part.

```
grade = 40

if grade > 65:
    print('You passed.')
    print('Congrats!')
elif grade > 55:
    print('You could not pass!')
    print('You can take the test one more time.')
else:
    print('You failed.')
    print('I am sorry.')
```

You failed.  
I am sorry.

## Nested if statements

[Skip to main content](#)

```
age = 10
weight = 45

if age > 5:                                # True: execute the block code within this `if` statement
    if weight>50:                            # False: skipped
        print('A')
    else:
        print('B')                           # executed
```

B

```
age = 10
weight = 55

if age > 5:                                # True: execute the block code within this `if` statement
    if weight>50:                            # True: executed
        print('A')
    else:
        print('B')                           # skipped
```

A

```
age = 3
weight = 45

if age > 5:                                # False: skip the block code within this `if` statement
    if weight>50:
        print('A')
    else:
        print('B')
else:                                         # this 'else' statement corresponds to the first 'if' statement
    print('C')                           # executed
```

C

## Boolean Operators

Boolean operators include `and`, `or`, and `not`.

[Skip to main content](#)

- `not` is the negation operator.
- `&`, `|` can be used instead of `and`, `or` respectively.

They are also called logical operators and works as follows:

- The and operator returns True if both operands are True, otherwise, it returns False.
- The or operator returns True if at least one of the operands is True. It returns False only if both operands are False.
- The not operator returns the opposite boolean value of the operand. If the operand is True, not returns False, and vice versa.

These operators, also called logical operators, work as follows:

- The `and` operator returns True if both operands are True; otherwise, it returns False.
- The `or` operator returns True if at least one of the operands is True; it returns False only if both operands are False.
- The 'not' operator returns the opposite boolean value of the operand. If the operand is True, 'not' returns False, and vice versa."

<b>Value</b>	<b>Operator</b>	<b>Value</b>	=	<b>Result</b>
True	and	True	=	True
True	and	False	=	False
False	and	True	=	False
False	and	False	=	False

<b>Value</b>	<b>Operator</b>	<b>Value</b>	=	<b>Result</b>
True	or	True	=	True
True	or	False	=	True
False	or	True	=	True
False	or	False	=	False

[Skip to main content](#)

- `not False = True`

```
print(True and False)
```

False

```
print(True & False) # and
```

False

```
print(True or False)
```

True

```
print(True | False) # or
```

True

```
print(not True)
```

False

```
print(not False)
```

True

```
# True and True = True  
print( (3 > 1) & (10 > 8) )
```

[Skip to main content](#)

True

```
# False or True = True  
print( 7 == 8 ) | (10 > 8) )
```

True

```
# not True = False  
print( not ('arm'<'kite') )
```

False

```
# not False = True  
print( not ( 'a' in 'Apple' ) ) # a is not in Apple
```

True

## Example-1

- If the (weather is nice) or (I have \$5), then I will buy an ice cream.
  - If either of the conditions is True, I will buy an ice cream.
  - If both of the conditions are False, I will not buy an ice cream.
- So, we have the following scenarios based on weather conditions and the amount of money available:
  - (weather is nice) = True or (I have \$5) = True —> buy an ice cream
  - (weather is nice) = False or (I have \$5) = True —> buy an ice cream
  - (weather is nice) = True or (I have \$5) = False —> buy an ice cream
  - (weather is nice) = False or (I have \$5) = False —> do NOT buy an ice cream

## Example-2

- If the (weather is nice) and (I have \$5), then I will buy an ice cream

[Skip to main content](#)

- If either of the conditions is False, then I will not buy an ice cream.
- So we have the following cases depending on the weather conditions and money amount:
  - (weather is nice) = True and (I have \$5) = True —> buy an ice cream
  - (weather is nice) = False and (I have \$5) = True —> do NOT buy an ice cream
  - (weather is nice) = True and (I have \$5) = False —> do NOT buy an ice cream
  - (weather is nice) = False and (I have \$5) = False —> do NOT buy an ice cream

```
temperature = 100
money = 3

print(f'Temperature is {temperature}.')
print(f'I have ${money}.')

if temperature > 75 and money > 5:      # True and False = False
    print('Go outside!')                  # skipped
else:
    print('Stay at home!')                # executed
```

Temperature is 100.  
 I have \$3.  
 Stay at home!

```
temperature = 100
money = 10

print(f'Temperature is {temperature}.')
print(f'I have ${money}.')

if temperature > 75 and money > 5:      # True and True = True
    print('Go outside!')                  # executed
else:
    print('Stay at home!')                # skipped
```

Temperature is 100.  
 I have \$10.  
 Go outside!

## try and except

[Skip to main content](#)

It is similar to an if-else statement. If there is an error in the code, the entire program will be terminated. To prevent termination in the presence of errors, a try-except statement is often used.

- This kind of situation is very common, especially when a user enters input that is not appropriate.
  - entering "one" instead of digit "1"
  - making typos like "5s" instead of "5"
- If you try to convert such inputs to a number, an error will occur, and the entire program may be terminated.
- To handle such situations gracefully, you can use a try-except statement
- **try-except** works as follows:
  - If there is no error in the block code of the try part, this block code will be executed.
  - If there is an error in the block code of the try part, the block code of the except part will be executed.
- The structure of a try-except statement is as follows:

**try:**



**except:**



- The cases are as follows:

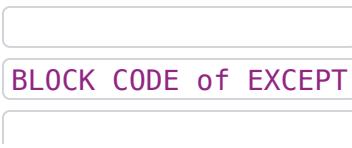
1. BLOCK CODE of TRY has no error:

**try:**



BLOCK CODE of TRY will be executed.

**except:**



BLOCK CODE of EXCEPT will be skipped.

[Skip to main content](#)

## 2. BLOCK CODE of TRY has an error:

try:

  BLOCK CODE of TRY

except:

  BLOCK CODE of EXCEPT

BLOCK CODE of TRY will be skipped.

BLOCK CODE of EXCEPT will be executed.

```
num = '5'  
try:  
    x = int(num)**2  
    print(f'Square of {num} is {x}')  
# no error: '5' can be converted to  
# executed  
except:  
    print('Warning: Please enter an integer.')  
# skipped
```

Square of 5 is 25

```
num = '5s'  
try:  
    x = int(num)**2  
    print(f'Square of {num} is {x}')  
# error: '5s' can not be converted  
# skipped  
except:  
    print('Warning: Please enter an integer.')  
# executed
```

Warning: Please enter an integer.

### Remark

- You must include the except part along with some code when using the try statement.
- If you don't intend to perform any specific actions in the except part, you can use the `pass` keyword to prevent an error.

```
num = '5s'  
try:  
    x = int(num)**2  
    print(f'Square of {num} is {x}')  
except:  
    pass
```

# error: '5s' can not be converted  
# skipped  
# does not do anything

No output

## Examples

### Even or Odd

- Ask for an integer from the user and check whether it is even or odd.
- Print your result using f-strings in the form of "The given number is even/odd."

### Solution

```
number = int( input('Enter an integer:') )  
  
if number%2 == 0:                                # for even numbers, the remainder is zero,  
    print(f'{number} is an even number')  
else:  
    print(f'{number} is an odd number')
```

### Greater than ten

- Ask the user for an integer and check whether it is greater than 10 or not.
- Print the result using f-strings in the form: "The given number is greater/not greater than 10."

### Solution

```
number = int(input('Enter an integer:'))  
  
if number>10 :                                # this condition is True if the number is g  
    print(f'{number} is greater than 10')  
_____
```

[Skip to main content](#)

## Same names

- Ask for two names from the user using two input() functions and check whether these names are the same.
- It should not be case-sensitive, meaning that "Tom" and "TOM" are considered the same name.
- Print the result using f-strings in the form of "{name1}" and "{name2}" are the same/not the same.
- Print the names exactly as given by the user.

### Solution

```
name1 = input('Please enter the first name: ')
name2 = input('Please enter the second name: ')

if name1.lower() == name2.lower():          # compare lower case versions to make it
    print(f'{name1} and {name2} are same.')
else:
    print(f'{name1} and {name2} are not same.')
```

## Letter Grades

- Write a program that asks the user to enter a percent grade.
  - Display the corresponding letter grade according to the following chart.
- | Letter Grade | Grade Range |
|--------------|-------------|
| A            | 80 - 100    |
| B            | 60 - 79     |
| C            | 40 - 59     |
| D            | 20 - 39     |
| F            | 0 - 19      |

### Solution

```
grade = float( input( 'Enter your percent grade:' ) )

if 80 <= grade <= 100:
    print('Your letter grade is A')
elif 60 <= grade :
    print('Your letter grade is B')
elif 40 <= grade :
    print('Your letter grade is C')
elif 20 <= grade :
    print('Your letter grade is D')
elif 0 <= grade :
    print('Your letter grade is F')
else:
```

[Skip to main content](#)

# Piecewise Defined Function

The piecewise-defined function  $f(n)$  is given as follows:

$$f(n) = \begin{cases} 4 - 2n & n < -2 \\ 5 & -2 \leq n \leq 7 \\ 1 - n & n > 7 \end{cases}$$

- Write a program that asks the user to enter an integer.
- If the integer entered is  $n$  then display  $f(n)$ .
- Hint:
  - If  $n$  is less than  $-2$  then  $f(n)=4-2n$
  - If  $n$  is greater than or equal to  $-2$  and less than or equal to  $7$  then  $f(n)=5$
  - If  $n$  is greater than  $7$  then  $f(n)=1-n$
- Example:
  - $f(-4) = 4 - 2(-4) = 12$  since  $-4 < -2$
  - $f(1) = 5$  since  $-2 \leq 1 \leq 7$
  - $f(10) = 1 - 10 = -9$  since  $10 > 7$

## Solution

```
n = int(input('Please enter a number:'))\n\nif n < -2:\n    print(f'f({n})={4-2*n}')\nelif -2 <= n <= 7:\n    print(f'f({n})=5')\nelse:\n    print(f'f({n})={1-n}')
```

# Secret Number Game

- Choose a random integer between 1 and 10 as the secret number.
- Ask a number from the user to guess it.
- If the user's guess is correct, display You win!
- If the user's guess is incorrect, display Incorrect. Try again!!

[Skip to main content](#)

- If the user’s guess is 99999, display You win!
- Use try and except to avoid errors if the user enters non-numeric values.
- Warn the user if there is an error by displaying a message.

## Solution

```
import random
secret_number = random.randint(1,10)

try:
    player = int(input('Guess the number: '))

    if player == secret_number:
        print('You win!')
    elif player == 99999:      # cheating part
        print('You win!')
    else:
        print('Incorrect. Try Again!')

except:
    print('Please enter a valid numeric value!')
```

## Conditionals Debugging

- Each of the following short code contains one or more bugs.
- Please identify and correct these bugs.
- Provide an explanation for your answer.

## Question

```
x = 10

if x<20:
print('A')
```

### Solution

Add appropriate indentation to the last line.

[Skip to main content](#)

## Question

```
x = 10  
if x<20  
    print(x+5)
```

### Solution

Colon (:) is missing at the end of the second line.

## Question

```
x = 10  
if x<20:  
    print(A)
```

### Solution

A is not defined.

## Question

```
is_cheap = true  
if is_cheap:  
    print('Buy it')
```

### Solution

In the first line, true should be capitalized as True to represent the boolean value.

## Question

[Skip to main content](#)

```
house_age = 20

if house_age > 20:
    print('OLD')
elif house_age = 20:
    print('Twenty')
else:
    print('NEW')
```

### Solution

In the condition of the elif part, replace = with == to form a correct boolean expression.

## Question

```
x = 5

if x > 10:
    print('A')
elif:
    print('B')
else:
    print('C')
```

### Solution

The condition (boolean expression) of the elif part is missing.

## Conditionals Output

- Find the output of the following code.
- Please don't run the code before giving your answer.

## Question

```
print(True+False+3+True)
```

[Skip to main content](#)

## Question

```
x = 20
y = 100

b1 = x == y
b2 = x < y

print(b1)
print(b2)
print(b1 and b2)
print(b1 or b2)
```

► Show code cell output

## Question

```
x = 10

if x<7:
    print('A')
```

- No output!

## Question

```
x = 20

if x > 10:
    print('A')
elif x > 15:
    print('B')
else:
    print('C')
```

► Show code cell output

## Question

[Skip to main content](#)

```
x = 20  
  
if x > 10:  
    print('A')  
if x > 15:  
    print('B')
```

► Show code cell output

## Question

```
x = 17  
y = 9  
  
if x<15 and y>5:  
    print('A')  
elif x>7 and y>5:  
    print('B')  
elif x>10 and y>6:  
    print('C')  
else:  
    print('D')
```

► Show code cell output

## Question

```
x = 25  
  
if x < 30:  
    print('A')  
    if x < 20:  
        print('B')
```

► Show code cell output

## Question

[Skip to main content](#)

```
x = 25  
  
if x < 30:  
    print('A')  
    if x < 40:  
        print('B')
```

► Show code cell output

## Question

```
x = 25  
  
if x < 10:  
    print('A')  
    if x < 40:  
        print('B')
```

- No output!

## Question

```
if 10:  
    print('USA')
```

► Show code cell output

## Question

```
if 0:  
    print('USA')
```

- No output!

## Question

[Skip to main content](#)

```
if 0==0 :  
    print('USA')
```

► Show code cell output

## Question

```
x = 10  
  
if x > 2*x:  
    print(x)  
print(2*x)
```

► Show code cell output

## Question

```
x = 10  
  
if x > 2*x:  
    print(x)  
print(2*x)
```

- No output!

## Question

```
if True:  
    if 1:  
        if 5:  
            if 0:  
                print('A')  
            else:  
                print('B')  
        else:  
            print('C')
```

► Show code cell output

[Skip to main content](#)

```
x = 5

if x > 3:
    print('A')
if x > 4:
    print('B')
    print('***')
if x > 10:
    print('C')
    print('---')
print('D')
```

► Show code cell output

## Question

```
x = '3'

try:
    print(x+5)
except:
    print('ERROR')
```

► Show code cell output

## Question

```
x = True

try:
    print(x+5)
except:
    print('ERROR')
```

► Show code cell output

## Conditionals Code

- Please solve the following questions using Python code.

[Skip to main content](#)

Write a program that asks the user to enter a 6-digit number.

- Check whether the digit in the thousands place is even.

## Solution

### Solution

```
number = input('Please enter a 6 digit number: ')
thousands_digit = int(number[2])

if thousands_digit%2:
    print(f'Thousands digit is odd.')
else:
    print(f'Thousands digit is even.')
```



## Question

Write a program that asks the user to enter a percent grade.

- Display the corresponding “Letter Grade” according to the following grading scale.

Letter Grade	Grade Range
A	96 - 100
A-	90 - 95
B+	87 - 89
B	84 - 86
B-	80 - 83
C+	77 - 79
C	74 - 76
C-	70 - 73
D+	67 - 69
D	64 - 66
D-	60 - 63
F	0 - 59

## Solution

### Solution

```
grade = float(input('Please enter your percent grade: '))

if grade >= 96: print('A')
elif grade >= 90: print('A-')
elif grade >= 87: print('B+')
elif grade >= 84: print('B')
elif grade >= 80: print('B-')
elif grade >= 77: print('C+')
elif grade >= 74: print('C')
elif grade >= 70: print('C-')
elif grade >= 67: print('D+')
elif grade >= 64: print('D')
elif grade >= 60: print('D-')
else: print('F')
```

[Skip to main content](#)

## Question

The piecewise-defined function  $f(n)$  is given as follows:

$$f(n) = \begin{cases} n^2 + 3n + 4 & n > 7 \\ 3n - 7 & 7 \geq n > 4 \\ 10 & 4 \geq n \geq 2 \\ 3 - 4n & \text{otherwise} \end{cases}$$

- Write a program that asks the user to enter an integer.
- If the integer entered is  $n$  then display  $f(n)$ .
- Hint:
  - If  $n$  is greater than 7 then  $f(n)=n^2+3n+4$ .
  - If  $n$  is between 4 (4 is not included) and 7 (7 is included) then  $f(n)=3n-7$ .
  - If  $n$  is between 2 (2 is not included) and 4 (4 is included) then  $f(n)=10$ .
  - If  $n$  is less than 2 then  $f(n)=3-4n$ .

## Solution

### Solution

```
n = int( input('Enter an integer: ') )  
  
if n > 7:  
    print(f'f(n) = {n**2+3*n+4}')  
elif 4 < n:  
    print(f'f(n) = {3*n-7}')  
elif 2 <= n:  
    print(f'f(n) = 10')  
else:  
    print(f'f({n}) = {3-4*n}')
```

## Question

Temperature Converter: Fahrenheit (F)  $\longleftrightarrow$  Celsius (C).

- Ask for temperature with unit (F or C) from the user.

[Skip to main content](#)

- The user should enter the temperature in the form (int)F or (int)C.
  - Example: 37F or 42C
- If the temperature is given in Fahrenheit (F) by the user, then convert it to Celsius (C).
- If the temperature is given in Celsius (C) by the user, then convert it to Fahrenheit (F).
- Display the converted temperature with its unit.

## Solution

### Solution

```
temperature_unit = input('Please enter the temperature with the unit: ')

temp = int(temperature_unit[:-1])          # the slice, except for the last character
unit = temperature_unit[-1]                 # the last character is the unit

if unit == 'F':
    print(f'It is {(temp-32)/1.8}C.')
else:
    print(f'It is {temp*1.8+32}F.')
```

## Question

Ask the user to input a non-negative integer.

- Check if this integer is a perfect square and print your conclusion.
  - Perfect squares are squares of integers, for example: 0, 1, 4, 9, 16, 25, 36, 49, etc.
- If the given number is a negative integer, print a warning message.
- Utilize the sqrt function from numpy or math.
- Example:
  - For number = -3, output: -3 < 0. Please enter a non-negative integer.
  - For number = 4, output: 4 is a perfect square.
  - For number = 12, output: 12 is NOT a perfect square.

## Solution

[Skip to main content](#)

## Solution

```
import math
number = int(input('Enter a non-negative integer: '))

if number < 0:
    print(f'{number} < 0. Please enter a non-negative integer.')
else:
    int_sqrt = int(math.sqrt(number))           # if a number is a perfect square
    if int_sqrt**2 == number:                   # so the square root's integer pa
        print(f'{number} is a perfect square.')
    else:
        print(f'{number} is a NOT perfect square.)
```

## Question

Ask for a name from the user.

- Check if it contains the letters 'k' or 'K' and if the length of the name is an odd number.
- If this is the case, then replace 'k' with 'y' and 'K' with 'Y'.
- Display the new name.
- Example:
  - name = Tom Output: Tom
  - name = Jack Output: Jack (length is even)
  - name = KaThy Output: YaThy
  - name = katHy Output: yatHy

## Solution

### Solution

```
name = input('Enter a name: ')

if ('k' in name.lower()) & (len(name)%2==1):
    print(name.replace('k', 'y').replace('K', 'Y'))
else:
    print(name)
```

[Skip to main content](#)

## Question

Ask for a number from the user

- Check whether the given number satisfies the following inequality:  $x^2 - 3x + 6 > 192$
- Display the conclusion.

### Solution-1

#### Solution

```
x = float(input('Enter a number: '))
y = x**2-3*x+6
if y > 192:
    print(f'{x} satisfies the inequality')
else:
    print(f'{x} does NOT satisfy the inequality')
```

### Solution-2

#### Solution

```
x = float(input('Enter a number: '))
inequality = (x**2-3*x+6) > 192
if inequality:
    print(f'{x} satisfies the inequality')
else:
    print(f'{x} does NOT satisfy the inequality')
```

## Question

Ask for Test-1, Test-2, and Final exam grades from the user.

- Use three input functions.
  - Compute the weighted average by using the following formula.

[Skip to main content](#)

- Weighted Average =  $0.2 \times \text{Test-1} + 0.3 \times \text{Test-2} + 0.5 \times \text{Test-3}$
- Find the letter grade by using the following grading scale.

Weighted Average	Letter Grade
75-100	A
60-74	B
40-59	C
0-39	F

- Example:
  - Test-1 grade: 70
  - Test-2 grade: 80
  - Final grade: 90
  - Weighted average =  $0.2 \times 70 + 0.3 \times 80 + 0.5 \times 90 = 14 + 24 + 45 = 83$
  - Output:
    - Weighted Average 83.0 —> Letter Grade: A

## Solution

### Solution

```
test1 = float(input('Test-1 grade: '))
test2 = float(input('Test-2 grade: '))
final = float(input('Final grade: '))

weighted_grade = test1*0.2+test2*0.3+final*0.5

print(f'Weighted Average {weighted_grade} ----> Letter Grade: ', end='')

if 100 >= weighted_grade >= 75:
    print('A')
elif weighted_grade >= 60:
    print('B')
elif weighted_grade >= 40:
    print('C')
else:
    print('D')
```

[Skip to main content](#)

## Question

- Print the statement: Basic Calculator
  - Print 20 dashes (-)
  - Ask the user for two numbers using two input functions
  - Ask the user for the operation to perform
    - Provide the options: +, -, \*, /
    - The user should choose one of them.
  - Find the result
    - Perform the operation: First Number operation Second Number.
    - If the second number is zero, do not perform the division operation and display a warning message.
    - If the operation is division, also round the result.
  - Example: num1=5, num2=7, operation=''
- *Output: 57=35*

## Solution

### Solution

```
print('Basic Calculator')
print('-'*20)

number1 = float(input('Number-1: '))
number2 = float(input('Number-2: '))
operation = input('Operation: (+,-,*,/): ')

if operation == '+':
    print(f'{number1} {operation} {number2} = {number1 + number2}')
elif operation == '-':
    print(f'{number1} {operation} {number2} = {number1 - number2}')
elif operation == '*':
    print(f'{number1} {operation} {number2} = {number1 * number2}')
else:
    if number2 == 0:
        print('Warning: Division by zero.')
    else:
        print(f'{number1} {operation} {number2} = {number1 / number2:.2f}')
```

## Question

[Skip to main content](#)

Ask the user for a number with at most 8 digits.

- If the given number has fewer than 8 digits, pad zeros to the left to make it an 8-digit number.
- Example:
  - Given number: 123 → Output: 00000123
  - Given number: 123456789 → Output: Please enter a number with at most 8 digits.
  - Given number: 12345678 → Output: 12345678

## Solution

### Solution

```
number = input('Please enter a number with at most 8 digits: ')

length = len(number)

if length <= 8:
    print((8-length)*'0'+number)
else:
    print('Please enter a number with at most 8 digits.')
```

## Chp-6: Iterations

- Learning Objectives
  - ..
  - ..

## Motivation

### Triangle

As a motivational exercise, let's recall the code we previously encountered for printing a triangle using the  character and the *print()* function.

[Skip to main content](#)

```
print('&')
print('&&')
print('&&&')
print('&&&&')
print('&&&&&')
print('&&&&&&')
print('&&&&&&&')
print('&&&&&&&&')
print('&&&&&&&&&')
```

&  
&&  
&&&  
&&&&  
&&&&&  
&&&&&&  
&&&&&&&  
&&&&&&&&  
&&&&&&&&&

This code follows a pattern where the number of & characters increases by one in each line, simplifying the code through repetition.

```
print('&*1')
print('&*2')
print('&*3')
print('&*4')
print('&*5')
print('&*6')
print('&*7')
print('&*8')
print('&*9')
print('&*10')
```

&  
&&  
&&&  
&&&&  
&&&&&  
&&&&&&  
&&&&&&&  
&&&&&&&&

[Skip to main content](#)

- The second version, utilizing string repetition, is simpler than the first.
- However, further simplification is possible as there are still repetitions, such as the presence of the `print()` function in each line.
- To address this, iterations can be employed to avoid using the `print()` function ten times.
- The iteration version is as follows:
  - It eliminates repetition.
  - It does not become longer even with a larger triangle.

```
for i in range(1,11):      # iteration
    print('&'*i)
```

&  
 &&  
 &&&  
 &&&&  
 &&&&&  
 &&&&&&  
 &&&&&&&  
 &&&&&&&&  
 &&&&&&&&&

## Strings

- We have explored various methods for working with strings, yet a crucial aspect remains unaddressed — how to iterate through all characters of a string individually.
- While indexes and slices allow access to specific characters or portions of a string, the challenge lies in accessing each character sequentially.

### 1. Question:

What is the occurrence of a certain character, such as 'r', in a string?

- How can we address this question without utilizing the `count()` method of strings?
- The solution involves checking whether each character in the string matches 'r'.
- Let's attempt to write code that answers this question using a short string and only the information we have gathered from the previous chapters.

```
text = 'radar'
count_r = 0

if text[0] == 'r':
    count_r += 1
if text[1] == 'r':
    count_r += 1
if text[2] == 'r':
    count_r += 1
if text[3] == 'r':
    count_r += 1
if text[4] == 'r':
    count_r += 1

print(f'There are {count_r} "r" characters in {text}.')
```

There are 2 "r" characters in radar.

- As evident, there are numerous repetitions in this code, making it overwhelming for long strings.
- To mitigate this, iterations can be employed to avoid the need for using the *if* statements multiple times.
- The iteration version is as follows, it eliminates repetition and does not become longer even with an extended string.

```
text = 'radar'
count_r = 0

for char in text:          # iteration
    if char == 'r':
        count_r += 1

print(f'There are {count_r} "r" characters in {text}.')
```

There are 2 "r" characters in radar.

2. **Question:** Now let's work on a more complicated question. What are the digits in a string which are greater than 6?

- Let's attempt to write code that answers this question using a string which consists of digits and only the information we have gathered from the previous chapters.

[Skip to main content](#)

```
text = '192736'
print(f'The digits in {text} which are greater than 6:')

if int(text[0]) > 6:
    print(text[0])
if int(text[1]) > 6:
    print(text[1])
if int(text[2]) > 6:
    print(text[2])
if int(text[3]) > 6:
    print(text[3])
if int(text[4]) > 6:
    print(text[4])
if int(text[5]) > 6:
    print(text[5])
```

The digits in 192736 which are greater than 6:

9  
7

- The code above exhibits repetition, which can be overwhelming for long strings.
- The iteration version, provided below, eliminates this repetition and does not become longer even with an extended string.

```
text = '192736'
print(f'The digits greater than 6 in {text}:')

for char in text:
    if int(char) > 6:
        print(char)
```

The digits greater than 6 in 192736:

9  
7

## Iterations

Iterations are used to perform the same or similar tasks in a more efficient way.

- Similar tasks usually follow a pattern that can be used to write the code in a shorter and more

[Skip to main content](#)

There are two types of iterations available in programming languages:

- `while` loop: This is used for indefinite repetition, executing a block code for a possibly unknown number of times.
- `for` loop: This is used for definite repetition, executing a code for a known number of times.

1. The `for` loop executes its block code repeatedly for every element of a sequence.
  - It has a condition (boolean expression) with the `in` operator, making it possible to execute the block code only for elements of the sequence.
2. The `while` loop executes a block code as long as its condition is True.
  - It is similar to an `if` statement because its condition can be any boolean expression (not only with `in`).
  - The difference lies in the fact that the block code of an `if` statement is executed only once if the condition is True.
  - In contrast, if the condition of the `while` loop is True, its block code is executed as in `if` statements, but then the condition is checked again.
  - If it is still True, the block code will be executed again. The `while` loop keeps executing its block code as long as its condition becomes False or a break command is used to terminate it.

## Range Function

The built-in `range()` function returns a sequence of integers.

- The type of its output is `range`, and its values are hidden within it.
- You can use the built-in `list()` function to explicitly display the numbers in a range type output.
- The `range()` function has three important parameters: start, end, and step.
  - How they work is similar to the start, end, and step used for slicing of strings by using indexes.
- There are three cases:  
|#|Function|Numbers|Explanation| |-|-|-|-  
|1|`range(a)` | 0, 1, 2, ..., a-1 | integers starting from 0 goes upto a-1  
|2|`range(a,b)` | a, a+1, ..., b-1 | integers starting from a goes upto b-1  
|3|`range(a,b,s)` | a, a+s, a+2s, ..., less than b-1 | integers start from a go upto b-1 with an increment of s|
- The step  $s$  can be a negative number. If  $s$  is a negative number:
  - If  $a < b$ , the output is empty (as you cannot reach  $a$  from  $b$  by adding negative numbers).  
~ If  $a > b$ , the output is  $a, a - 1, \dots, b + 1$  (so you can reach  $a$  from  $b$  by adding a negative

[Skip to main content](#)

- Example:

- range(10) consists of 0, 1, 2, 3, 4, 5, 6, 7, 8, 9.
- range(2,10) consists of 2, 3, 4, 5, 6, 7, 8, 9.
- range(2,10,3) consists of 2, 5, 8.
- range(2,10,-3) is empty.
- range(10,2,-3) consists of 10, 9, 8, 7, 6, 5, 4, 3.

```
# Case 1: range(a)
rng_numbers = range(10)

print(f'Output: {rng_numbers}')
print(f'Type : {type(rng_numbers)}')
print(f'List : {list(rng_numbers)})
```

```
Output: range(0, 10)
Type : <class 'range'>
List : [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

```
# Case 2: range(a,b)
rng_numbers = range(2,10)

print(f'Output: {rng_numbers}')
print(f'Type : {type(rng_numbers)}')
print(f'List : {list(rng_numbers)})
```

```
Output: range(2, 10)
Type : <class 'range'>
List : [2, 3, 4, 5, 6, 7, 8, 9]
```

```
# Case 3: range(a,b,s)
rng_numbers = range(2,10,3)

print(f'Output: {rng_numbers}')
print(f'Type : {type(rng_numbers)}')
print(f'List : {list(rng_numbers)})
```

```
Output: range(2, 10, 3)
Type : <class 'range'>
List : [2, 5, 8]
```

[Skip to main content](#)

```
# Case 3: a<b and negative s
rng_numbers = range(2,10,-3)

print(f'Output: {rng_numbers}')
print(f'Type : {type(rng_numbers)}')
print(f'List : {list(rng_numbers)')

list(range(10,2,-1))
```

```
Output: range(2, 10, -3)
Type : <class 'range'>
List : []
```

```
[10, 9, 8, 7, 6, 5, 4, 3]
```

```
# Case 3: a>b and negative s
rng_numbers = range(10,2,-3)

print(f'Output: {rng_numbers}')
print(f'Type : {type(rng_numbers)}')
print(f'List : {list(rng_numbers)')

list(range(10,2,-1))
```

```
Output: range(10, 2, -3)
Type : <class 'range'>
List : [10, 7, 4]
```

```
[10, 9, 8, 7, 6, 5, 4, 3]
```

## for loop

The structure of a `for` loop is as follows:

```
for i in sequence:
```

```
    BLOCK CODE
```

[Skip to main content](#)

- The outputs of the `range()` function and strings can be used as sequences.
  - Each number in the output of the `range()` function will be an  $i$  value.
  - Each character of the strings will be an  $i$  value.
- $i$  is the counter, and you can choose a different name for it.
- For each  $i$  value from the sequence, the block code will be executed.
- The output depends on what  $i$  does in the block code.

**Example:** In the code below the values of  $i$  are: 3, 4, 5.

- The `print` statement will be executed for each  $i$  value one by one.
- The squares of the  $i$  values will be printed.

iteration #	$i$	$i^2$
1	3	9
2	4	16
3	5	25

```
[for i in range(3,6):
    print(i**2)]
```

```
9
16
25
```

**Example:** In the code below the values of  $i$  are: 1, 2, 3, ..., 10

- In every iteration  $i$  many `&` characters are printed.

```
[for i in range(1,11):
    print('&'*i)]
```

```
&
&&
&&&
&&&&
&&&&&
&&&&&&
&&&&&&&
&&&&&&&&
```

**Example:** In the code below, the values of  $j$  are: 3, 4, 5.

- The block code (4 lines) will be executed for each  $j$  value one by one.
- The value of each *frac* variable is calculated as shown in the table below.
- The calculated *frac* values will be printed.

iteration #	j	$\text{num} = 3 \times j + 2$	$\text{den} = 10^j$	$\text{frac} = \text{num}/\text{den}$
1	3	$3 \times 3 + 2 = 11$	$10^3 = 1,000$	0.011
2	4	$3 \times 4 + 2 = 14$	$10^4 = 10,000$	0.0014
3	5	$3 \times 5 + 2 = 17$	$10^5 = 100,000$	0.00017

```
for j in range(3,6):
    num = 3*j+2
    den = 10**j
    frac = num/den
    print(frac)
```

```
0.011
0.0014
0.00017
```

**Example:** In the code below, the values of  $i$  are: 'u', 't', 'a', 'h'.

- In every iteration, the value of  $i$ , which is a character of 'utah' is printed.

```
for i in 'utah':
```

[Skip to main content](#)

u  
t  
a  
h

**Example:** In the code below, the values of *i* are: 'u', 't', 'a', 'h'.

- The condition of the *if* statement is True if the value of *i* is before 'k' in dictionary order.
  - This is False for 'u' and 't', so they are not printed.
  - This is True for 'a' and 'h', so they are printed.

```
for i in 'utah':  
    if i < 'k':  
        print(i)
```

a  
h

**Example:** In the code below, the values of *i* are: 3, 4, 5.

- The initial value of the *total* variable is 0.
- In each iteration, the value of *i* is added to the *total*.

iteration #	<i>i</i>	<i>total</i>
-	-	0
1	3	0+3=3
2	4	3+4=7
3	5	7+5=12

```
total = 0  
  
for i in range(3,6):  
    total += i  
    print(f'Iteration number:{i-2}    i:{i}--->total:{total}')
```

[Skip to main content](#)

```
Iteration number:1  i:3--->total:3
Iteration number:2  i:4--->total:7
Iteration number:3  i:5--->total:12
```

## break and continue

**break** is used to terminate the *for* loop.

- It is usually used in an *if* statement to terminate the *for* loop under certain conditions.

**continue** is used to skip the rest of the body code of the *for* loop.

- It goes back to the beginning of the loop.
- It does not terminate the loop, just skips the rest of the block code for that iteration.

**Example:** In the code below, the values of *i* are: 1, 2, 3, 4.

- for *i* = 1 and *i* = 2, the *if* part is not executed since its condition is False, and the values 1 and 2 are printed.
- for *i* = 3, **break** is executed, and the loop is terminated.

```
for i in range(1,5):
    if i == 3:
        break
    print(i)
```

```
1
2
```

**Example:** In the code below, the values of *i* are: 1, 2, 3, 4.

- For *i* = 1 and *i* = 2, the *if* part is not executed since its condition is False, and the values 1 and 2 are printed.
- For *i* = 3, **continue** is executed, and the print statement is skipped, and the *i* = 4 iteration is started.
- For *i* = 4, the *if* part is not executed since its condition is False, and the value 4 is printed

[Skip to main content](#)

```
for i in range(1,5):
    if i == 3:
        continue
    else:
        print(i)
```

1  
2  
4

## for and else

for loops can have an `else` statement.

- The `else` statement is executed when the `for` loop is completed without any `break`.

**Example:** In the code below, the values of  $i$  are: 1, 2, 3, 4.

- After executing the `print()` function for  $i = 4$ , the `for` loop is over, and the `else` part is executed.

```
for i in range(1,5):
    print(i)
else:
    print('Over')
```

1  
2  
3  
4  
Over

**Example:** In the code below, the values of  $i$  are: 1, 2, 3, 4.

- The condition of the `if` statement is True when  $i$  is 4, and the `break` is executed, so the `for` loop is terminated.
  - The `else` part is not executed.

```
for i in range(1,5):
    print(i)
    if i > 3:
        break
else:
    print('Over')
```

1  
2  
3  
4

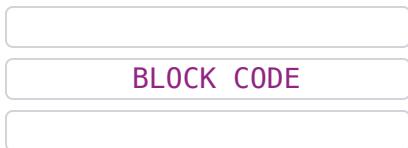
## while loop

The `while` loops are similar to the `if` statements. In `if` statements, block code is executed only once when the condition is True, whereas in `while` loops, the block code might be executed more than once.

- If the condition is True, the block code is executed as in `if` statements, but then the condition is checked again.
- If it is still True, the block code of the `while` loop is executed again.
- This process continues as long as the condition is True.
- Whenever the condition becomes False, the `while` loop is terminated.

The structure of a `while` loop is as follows:

`while condition:`



- `condition` is a Boolean expression (True or False)
- Possible conditions:
  - True, False;
  - <, >, <=, >=, ==, !=
  - not, and, or

[Skip to main content](#)

**Example:** In the code below, the initial value of  $n$  is 3.

1.  $n = 3$ : Check the condition.
  - Since  $3 > 0$ , the condition is True, and the block code is executed.
  - 3 is printed, and  $n$  becomes  $3 - 1 = 2$ .
2.  $n = 2$ : Check the condition.
  - Since  $2 > 0$ , the condition is True, and the block code is executed.
  - 2 is printed, and  $n$  becomes  $2 - 1 = 1$ .
3.  $n = 1$ : Check the condition.
  - Since  $1 > 0$ , the condition is True, and the block code is executed.
  - 1 is printed, and  $n$  becomes  $1 - 1 = 0$ .
4.  $n = 0$ : Check the condition.
  - Since  $0 > 0$  is False, the condition is False, and the loop is terminated.

```
n = 3  
while n>0:  
    print(n)  
    n -= 1
```

```
3  
2  
1
```

**Example:** In the code below, the initial value of  $n$  is 3.

1.  $n = 3$ : Check the condition.
  - Since  $bool(3)$  is True, the condition is True, and the block code is executed.
  - 3 is printed, and  $n$  becomes  $3 - 1 = 2$ .
2.  $n = 2$ : Check the condition.
  - Since  $bool(2)$  is True, the condition is True, and the block code is executed.
  - 2 is printed, and  $n$  becomes  $2 - 1 = 1$ .
3.  $n = 1$ : Check the condition.
  - Since  $bool(1)$  is True, the condition is True, and the block code is executed.

.. . . . . 1 1 0

[Skip to main content](#)

4.  $n = 0$ : Check the condition.

- Since  $\text{bool}(0)$  is False, the condition is False, and the *while* loop is terminated

```
n = 3  
  
while n:  
    print(n)  
    n -= 1
```

```
3  
2  
1
```

**Example:** In the code below, the initial values are set with  $n = 3$  and  $\text{total} = 0$ .

1.  $n = 3, \text{total} = 0$ : Check the condition.

- Since  $3 < 6$ , the condition is True, and the block code is executed.
- $\text{total} = 0 + 3 = 3$
- print Iteration number: 1  $n : 3 \rightarrow \text{total} : 3$
- $n = 3 + 1 = 4$

2.  $n = 4, \text{total} = 3$ : Check the condition.

- Since  $4 < 6$ , the condition is True, and the block code is executed.
- $\text{total} = 3 + 4 = 7$
- print Iteration number: 2  $n : 4 \rightarrow \text{total} : 7$
- $n = 4 + 1 = 5$

3.  $n = 5, \text{total} = 7$ : Check the condition.

- Since  $5 < 6$ , the condition is True, and the block code is executed.
- $\text{total} = 7 + 5 = 12$
- print Iteration number: 3  $n : 5 \rightarrow \text{total} : 12$
- $n = 5 + 1 = 6$

4.  $n = 6, \text{total} = 12$ : Check the condition.

- Since  $6 > 6$  is False, the condition is False, and the while loop is terminated.

iteration #	<i>n</i>	<i>total</i>
-	-	0
1	3	0+3=3
2	4	3+4=7
3	5	7+5=12

```
total = 0
n = 3
while n<6:
    total += n
    print(f'Iteration number:{n-2}    n:{n}--->total:{total}')
    n += 1
```

Iteration number:1 n:3--->total:3  
 Iteration number:2 n:4--->total:7  
 Iteration number:3 n:5--->total:12

## if versus while

In the following two examples, the same block code is used in an *if* statement and a *while* loop.

- In the *if* statement, the block code is executed only once for  $n = 3$ .
- In the *while* loop, the block code is executed two times for  $n = 3$  and  $n = 2$ .

**Example-1:** In the code below, the condition of the *if* statement is True, and the block code is executed.

- The print statement is executed, and the value of  $n$ , which is 3, is printed.
- Afterward,  $n$  is updated to  $3 - 1 = 2$ .
- The *if* statement is then concluded, and the only output is 3.

```
n = 3

if n > 1:
    print(n)
    n = n-1
```

[Skip to main content](#)

**Example-2:** In the code below, the initial value of  $n$  is 3.

1.  $n = 3$ : Check the condition.

- Since  $3 > 1$ , the condition is True, and the block code is executed.
- 3 is printed, and  $n$  becomes  $3 - 1 = 2$ .

2.  $n = 2$ : Check the condition.

- Since  $2 > 1$ , the condition is True, and the block code is executed.
- 2 is printed, and  $n$  becomes  $2 - 1 = 1$ .

3.  $n = 1$ : Check the condition.

- Since  $1 > 1$  is False, the condition is False, and the *while* loop is terminated.

```
n = 3
while n > 1:
    print(n)
    n = n-1
```

3  
2

## Infinite Loop

It is possible to have a condition for a *while* loop that is always True.

- In that case, the block code will be executed repeatedly unless the program is terminated by the user.
- This is not a syntax error, but it is not expected because the program will not end.

**Example:** In the code below, the initial value of  $n$  is 3, and in each iteration,  $n$  is increased by 1.

- Hence, the values of  $n$  are: 3, 4, 5, 6, ....
- The condition  $n > 1$  is always True, and the *while* loop never terminates.

```
n = 3

while n > 1:          # always True
    print(n)
    n = n+1           # n=3,4,5,.....
```

**Example:** In the code below, the condition is always True.

- 'Hello' is printed repeatedly, and the loop never terminates.

```
# infinite loop

while True:
    print('Hello')
```

## break and continue

They work similarly to the *for* loop. To prevent infinite loops, the use of `break` is particularly crucial for while loops.

**Example:** In the code below the initial value of *n* is 3.

1.  $n = 3$ : Check the condition.

- Since  $3 > 1$ , the condition is True, and the block code is executed.
- `3` is printed, and *n* becomes  $3 + 1 = 4$ .
- $3 \neq 5$ , so the condition of the if statement is False, and the `break` statement is skipped.

2.  $n = 4$ : Check the condition.

- Since  $4 > 1$ , the condition is True, and the block code is executed.
- `4` is printed, and *n* becomes  $4 + 1 = 5$ .
- $5 == 5$ , so the condition of the if statement is True, and the `break` statement is executed.
- The *while* loop is terminated.

```
n = 3

while n > 1:
    print(n)
    n = n+1
    if n == 5:
        break
```

[Skip to main content](#)

3  
4

## Examples

### Sum of Numbers

Find the sum of the numbers  $1, 2, 3, \dots, 100$  by using

1. a *for* loop
2. a *while* loop

3. The formula for the sum of the first  $n$  positive integers:  $1 + 2 + 3 + \dots + n = \frac{n(n + 1)}{2}$

### Solution

```
# for loop
total_for = 0
for i in range(1,101):
    total_for += i

# while loop
total_while = 0
n = 1
while n <= 100:
    total_while += n
    n +=1

#formula
total_formula = 100*(100+1)/2

print(f'for    loop answer: {total_for}')
print(f'while   loop answer: {total_while}')
print(f'formula  answer: {total_formula}')
```

```
for    loop answer: 5050
while   loop answer: 5050
formula  answer: 5050.0
```

[Skip to main content](#)

For the given text below, find the number of occurrences of the character 't' using:

1. a *for* loop
2. a *while* loop
3. a *string* method

```
text = """" Imyep jgsqewt okbxsq seunh many rkx vmysz ndpoz may vxabckewro topfd tqkj ue
```

## Solution

```
# for loop
count_for = 0
for char in text:
    if char == 't':
        count_for +=1

# while loop
count_while = 0
n = 0
while n < len(text):  # n values are the indexes
    if text[n] == 't':
        count_while +=1
    n += 1

# string method
count_method = text.count('t')

print(f'for    loop answer: {count_for}')
print(f'while   loop answer: {count_while}')
print(f'method  answer: {count_method}')
```

```
for    loop answer: 267
while   loop answer: 267
method  answer: 267
```

## Reverse a word

Write a program that reverses the given text below using:

1. a *for* loop
2. a *while* loop

[Skip to main content](#)

```
text = 'How are you?'
```

## Solution

```
# for loop
reverse_for = ''
for char in text:
    reverse_for = char + reverse_for

# while loop
reverse_while = ''
n = 0
while n < len(text):
    reverse_while = text[n] + reverse_while
    n += 1
# indexing
reverse_index = text[::-1]

print(f'for loop answer: {reverse_for}')
print(f'while loop answer: {reverse_while}')
print(f'indexing answer: {reverse_index}')
```

```
for loop answer: ?uoy era woH
while loop answer: ?uoy era woH
indexing answer: ?uoy era woH
```

## Secret Number Game

This is the updated version of the game in the conditionals chapter. In this new version, the user will keep guessing the number until quitting the game by entering 0.

- Choose a random integer between 1 and 10 as the secret number, and 0 to quit the game.
- Ask for a number from the user to guess the secret number.
  - If the user's guess is correct, display 'You win!'
  - If the user's guess is incorrect, display 'Incorrect. Try again!' and ask for a new guess.
  - If the user's input is 0, quit the game.
- Use *try* and *except* to avoid errors if the user enters non-numeric values.
  - Warn the user if there is an error by displaying a message.

[Skip to main content](#)

```

import random
secret_number = random.randint(1,10)      # choose a random number between 1 and 10

while True:                                # it will keep asking for a guess from the user

    try:
        player = int(input('Guess the secret number or press 0 to quit: '))

        if player == secret_number:
            print('Correct. You win!')
            break
        elif player == 0:
            print('Game is over!')
            break
        else:
            print('Incorrect. Try Again!')

    except:
        print('Please enter a valid numeric value!')

```

### Sample Output:

Guess the secret number or press 0 to quit: 6

Incorrect. Try Again!

Guess the secret number or press 0 to quit: 3

Incorrect. Try Again!

Guess the secret number or press 0 to quit: 8

Correct. You win!

## Factors-1

Write a program that asks the user to enter a positive integer.

- Display the positive factors of this number.
- A factor of a number is a positive integer that divides the given number without leaving a remainder.
- Example: The factors of 12 are 1, 2, 3, 4, 6, 12.

### Solution

```

n = int(  input('Enter a positive integer n:')  )
print(f'Factors of {n} are: ', end = '')

for i in range(1,n+1):      # factors are between 1 and the given number n

```

[Skip to main content](#)

```
print('\b') # remove the last comma
```

### Sample Output:

Enter a positive integer n: 12

Factors of 12 are: 1,2,3,4,6,12

## Factors-2

Write a program that asks the user to enter a positive integer.

- Display whether the numbers between 1 and the given number are positive factors of the given number.

### Solution

```
number = int( input('Enter a positive integer n:'))  
  
for i in range(1, number+1): # 1,2,3,4,5,6  
    if number % i == 0:      # i is a factor  
        print(f'{i} is a factor of {number}.')  
    else:                   # i is not a factor  
        print(f'{i} is NOT a factor of {number}.')
```

### Sample Output:

Enter a positive integer n: 12

1 is a factor of 12.

2 is a factor of 12.

3 is a factor of 12.

4 is a factor of 12.

5 is NOT a factor of 12.

6 is a factor of 12.

7 is NOT a factor of 12.

8 is NOT a factor of 12.

9 is NOT a factor of 12.

10 is NOT a factor of 12.

11 is NOT a factor of 12.

12 is a factor of 12.

[Skip to main content](#)

## Countdown

Write a program that counts down from 3 to 0 and displays these numbers with 'START' at the end.

- Add one second between each output by using the `sleep()` method of the `time` module.
  - `time.sleep(1)` delays the execution for 1 second.

### Solution

```
import time

for i in range(3,-1,-1):
    print(i)
    time.sleep(1)

print('START')
```

### Output:

```
3
2
1
0
START
```

## Count digits

Write a program which prints the digits which are greater than 6 in given string which might include any character.

- Use a `for` loop and `try-except`.

```
text = 'a9b4dh6e1_***8371__dthYFR8G12po7+'
```

### Solution:

```
print(f'The digits greater than 6 in {text}:')

for char in text:
    try:
        if int(char) > 6:
            print(char)
    except:
        pass
```

```
The digits greater than 6 in a9b4dh6e1_***8371__dthYFR8G12po7+:
9
8
7
8
7
```

## Iterations Debugging

- Each of the following short code contains one or more bugs.
- Please identify and correct these bugs.
- Provide an explanation for your answer.

### Question

```
for i range(6):
    print(i)
```

#### Solution

*in* in the first line is missing.

### Question

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

[Skip to main content](#)

## Solution

Indentation of the second line is missing.

## Question

```
while i in range(10):  
    print(i)
```

## Solution

*i* is undefined, or a *for* loop can be used instead of a *while* loop.

## Question

```
i = 5  
  
while i in range(10):  
    print(i)
```

## Solution

Infinite loop: You can increase the values of *i* by adding 1 after the print statement using *i += 1*.

## Question

```
x = 3  
  
while x > 0:  
    print(5*x)  
    x += 1
```

[Skip to main content](#)

### Solution

Infinite loop:  $x$  is continually increasing, ensuring it remains positive, and consequently, the condition of the *while* loop is always True. To prevent this, you can either decrease the values of  $x$  or add a *break* statement.

## Question

```
x = 1  
  
for x < 5:  
    print(x)  
    x += 1
```

### Solution

Instead of a *for* loop use a *while* loop.

## Iterations Output

- Find the output of the following code.
- Please don't run the code before giving your answer.

## Question

```
for i in range(13,29,5):  
    print(i)
```

► Show code cell output

## Question

```
for i in range(13,1,-3):  
    print(i)
```

[Skip to main content](#)

► Show code cell output

## Question

```
n = 5  
  
while n >= -3:  
    print(n)  
    n -= 2
```

► Show code cell output

## Question

```
for i in 'TEXAS':  
    print(i, end='-')
```

► Show code cell output

## Question

```
country = 'Germany'  
  
for i in range(2,len(country)):  
    print(country[:i])
```

► Show code cell output

## Question

```
text = 'abcdefghijklmnopqrstuvwxyz'  
  
for i in range(2,9,3):  
    print(text[i])
```

► Show code cell output

[Skip to main content](#)

```
text = 'abcdefghijklmnopqrstuvwxyz'

for i in text:
    if i >= 'w':
        print(i)
```

► Show code cell output

## Question

```
while True:
    print(1)
    if 5:
        break
```

► Show code cell output

## Question

```
x = 16

while x >= 1:
    print(x)
    x /= 2
print('DONE!')
```

► Show code cell output

## Question

```
total = 0

for i in range(2, 45, 10):
    total += i

print(total)
```

► Show code cell output

..

[Skip to main content](#)

```
for i in range(5):
    for j in range(6,10):
        print(j, end=' ')
    print()
```

► Show code cell output

## Question

```
for i in range(5):
    for j in range(i,3):
        print(j, end=' ')
    print()
```

► Show code cell output

## Question

```
for i in range(5,10):
    for j in range(4,i):
        print(j, end=' ')
    print()
```

► Show code cell output

## Iterations Code

- Please solve the following questions using Python code.

## Question

Using only one *for* loop and one *print()* function, display the following triangle.

[Skip to main content](#)

```
*  
****  
*****  
*****  
*****  
*****  
*****
```

## Solution

```
for i in range(1,20,3):  
    print(i*'*')
```

## Question

Find the sum of the squares of the following numbers in two different ways:  
3, 7, 11, 15, 19, 23, ..., 107.

- Use a *for* loop.
- Use a *while* loop.

## Solution

▶ Show code cell content

## Question

Write a program that asks the user to enter integers until the sum of the given integers exceeds 100.

- Display the sum and count of the entered numbers.
- Use a *while* loop.

[Skip to main content](#)

## Solution

```
total = 0
count = 0

while total < 100:
    number = int(input('Enter an integer: '))
    total += number
    count += 1

print(f'Sum = {total}, Count = {count}')
```

## Sample Output

Enter an integer: 3

Enter an integer: 90

Enter an integer: 6

Enter an integer: 10

<----->

Sum = 109, Count = 4

## Question

Find the following product using a *for* loop and round the final answer to the nearest hundredth.

$$\begin{array}{r} \cdot \quad 10 \quad 90 \quad 89 \quad 88 \quad 87 \quad 86 \quad 85 \quad 84 \\ \hline 100 \quad 99 \quad 98 \quad 97 \quad 96 \quad 95 \quad 94 \quad 93 \end{array}$$

## Solution

▶ Show code cell content

## Question

Write a program that displays a rectangle using the characters  and  (space).

- The rectangle has *width* many  characters on its upper and lower sides.
- The rectangle has *length* many  characters on its left and right sides.



[Skip to main content](#)

Width = 6 Height = 8	Width = 6 Height = 3	Width = 8 Height = 5
<pre>* * * * * * *           * *           * *           * *           * *           * *           * *           * * * * * * *</pre>	<pre>* * * * * * *           * *           * *           * *           * *           *</pre>	<pre>* * * * * * * * *           * *           * *           * *           * *           *</pre>

## Solution

```
# use the following variables
width, height = 8, 5
```

▶ Show code cell content

## Question

Write a program that displays a wide, one-floor building using the characters `*` and `' '` (space).

- The building consists of *room* many rectangles, each with a size of *width* by *height*, stacked horizontally.
- Some examples are as follows:

[Skip to main content](#)

Width = 4 Height = 6 Floor = 3	Width = 5 Height = 8 Floor = 4
<pre>* *</pre>	<pre>* *</pre>

## Solution

```
# use the following variables
width, height, room = 4, 6, 10
```

▶ Show code cell content

## Question

Write a program that displays a tall building using the characters  and  (space).

- The building consists of *floor* many rectangles, each with a size of *width* by *height*, stacked vertically.
- Some examples are as follows:

[Skip to main content](#)

Width = 10 Height = 5 Floor = 4	Width = 10 Height = 5 Floor = 6	Width = 5 Height = 6 Floor = 2	Width = 5 Height = 4 Floor = 3
<pre>* * * * * * * * * * *                               * *                               * *                               * *                               * * * * * * * * * * * *                               * *                               * *                               * *                               * * * * * * * * * * * *                               * *                               * *                               * *                               * * * * * * * * * * * *                               * *                               * *                               * *                               * * * * * * * * * * * *                               * *                               * *                               * *                               * * * * * * * * * * *</pre>	<pre>* * * * * * * * * * *                               * *                               * *                               * *                               * * * * * * * * * * * *                               * *                               * *                               * *                               * * * * * * * * * * * *                               * *                               * *                               * *                               * * * * * * * * * * * *                               * *                               * *                               * *                               * * * * * * * * * * *</pre>	<pre>* * * * * *           * *           * *           * *           * * * * * * *           * *           * *           * *           * * * * * * *           * *           * *           * *           * * * * * *</pre>	<pre>* * * * * *           * *           * *           * *           * * * * * * *           * *           * *           * *           * * * * * *</pre>

## Solution

```
# use the following variables
width, height, floor = 5, 6, 2
```

► Show code cell content

## Question

Find the sum of the first 1000 terms of the following sequence:

$$\frac{1}{1 \times 2}, \frac{1}{2 \times 3}, \frac{1}{3 \times 4}, \frac{1}{4 \times 5}, \dots$$

## Solution

► Show code cell content

## Question

Write a program that prompts the user to enter any text, which may include characters such as digits

[Skip to main content](#)

- Find the number of alphabet characters (a-z) in the given string.
- You can use the constant `ascii_letters` from the `string` module to access all lowercase and uppercase alphabet letters.
- Example:
  - Enter a string: Wer34
  - There are 3 alphabet letters in Wer34.

## Solution

```
import string

text = input('Enter a text: ')
count = 0

for i in text:
    if i in string.ascii_letters:
        count += 1

print(f'There are {count} alphabet letters in {text}')
```

## Sample Output

Enter a string: sD12&

There are 2 alphabet letters in sD12&

## Question

Write a program that generates a random word with 5 characters using lowercase alphabet letters.

- You can use `random.choice()` to randomly choose a letter.
- The generated word does not have to be meaningful.

## Solution

▶ Show code cell content

## Question

[Skip to main content](#)

- Find the number of zeroes at the end of the given number.
- Use a while loop."
- Example:
  - Input: 1234500 —> Output: 2

### Solution-1

```
number = int(input('Enter an integer: '))
count = 0

n = number
while n%10 == 0:
    count += 1
    n /= 10

print(f'There are {count} zeroes at the end of {number}.')
```

### Sample Output

Enter an integer: 278140000000000

There are 10 zeroes at the end of 278140000000000.

### Solution-2

```
number = input('Enter an integer: ')
count = 0

i = -1
while number[i] == '0':
    count += 1
    i -= 1

print(f'There are {count} zeroes at the end of {number}.')
```

### Sample Output

Enter an integer: 278140000000000

There are 10 zeroes at the end of 278140000000000.

Write a program that selects a 3-digit random number (dividend) and a 1-digit random number (divisor).

- After 10 seconds, display the remainder and quotient.
- Use `random.randint()` to generate random integers.

### Solution

```
import random
import time

divisor = random.randint(1,9)
dividend = random.randint(100,999)
print(f'Divide {dividend} by {divisor}')

time.sleep(10)

print(f'Quotient : {dividend//divisor}')
print(f'Remainder : {dividend%divisor}')
```

### Sample Output

Divide 495 by 4

Quotient : 123

Remainder : 3

## Chp-7: Tuples

- Learning Objectives
  - ..
  - ..

## Data Structures

Data structures are used to store data. We have already seen some of them including integers, floats, strings and booleans.

- By using these structures only one value can be stored.

[Skip to main content](#)

- 3.14 is a float with a single value.
- A string can be very long (have many characters) but still it is single value like 'Hello'.
- Boolean values are either *True* or *False* which are again single values.
- Integers, floats, strings and booleans are examples of **Primitive Data Structures** which means they can store only one value.

In Python, there are many more complicated data structures called **Imprimitive Data Structures** which can store more than one values with mixed types and have variuos functionalties including indexing. In this chapter, the imprimitive data structures **Tuples** will be covered.

Data structures are used to store data. We have already seen some of them, including integers, floats, strings, and booleans.

- By using these structures, only one value can be stored.
  - 3 is an integer and represents a single value.
  - 3.14 is a float with a single value.
  - A string can be very long (containing many characters), but it is still a single value, such as 'Hello'.
  - Boolean values are either True or False, again representing single values.
- Integers, floats, strings, and booleans are examples of **Primitive Data Structures**, meaning they can store only one value.

In Python, there are more complex data structures called **Non-Primitive Data Structures**, which can store multiple values with mixed types and have various functionalities, including indexing.

- In this chapter, the non-primitive data structure Tuples will be covered.
- Data structures
  1. Primitive
    - Integers
    - Floats
    - Strings
    - Booleans
  2. Impritive
    - Tuples

- Sets
- Arrays
- Dictionaries

## Tuples

Tuples are ordered sequences of values of mixed types.

- Since a tuple is ordered, the order of its elements is important.
  - 1, 2 and 2, 1 are different.
  - Since there is an order, indexing also works for tuples.
    - Indexing of tuples is very similar to strings.
- Values in a `tuple` can be of mixed types.
  - Integers, floats, strings, booleans, tuples, and lists can be values in a tuple.
- The built-in `tuple()` function can be used to convert appropriate data types into a tuple.
- `Tuples` are immutable (cannot be modified).
  - This is the main difference between tuples and lists.
  - Lists can be modified and will be covered in the upcoming chapter.
- The advantages of being immutable are:
  - Values cannot be modified, ensuring data protection.
  - Immutability makes tuples simpler, leading to faster and more memory-efficient operations.

## Create Tuples

A tuple can be created using one of the following methods:

1. Using a comma-separated sequence: 1, 2, 3.
  2. Enclosing a comma-separated sequence in parentheses: (1, 2, 3).
  3. Using the built-in `tuple()` function: `tuple(1, 2, 3)`.
- An empty tuple is represented by `( )`.
  - If a tuple has only one value, a comma should be added right after that single value.
    - Example: `1, or (1)`

[Skip to main content](#)

- `(1)` is the integer `1`, and `(1,)` is the tuple with only one value.
- `1` is the integer `1`, and `(1,`) is the tuple with only one value, which is `1`.

## Examples

```
# empty tuple
empty_tuple = ()

print(type(empty_tuple))
```

```
<class 'tuple'>
```

```
# empty tuple with tuple()
empty_tuple = tuple()

print(type(empty_tuple))
```

```
<class 'tuple'>
```

```
# tuple with only one value: 'USA'
t = 'USA',           # no parenthesis
print(type(t))
```

```
<class 'tuple'>
```

```
# tuple with only one value: 'USA'
t = ('USA',)         # with parenthesis
print(type(t))
```

```
<class 'tuple'>
```

```
# (1) is an integer not tuple
t = (1)
```

[Skip to main content](#)

```
<class 'int'>
```

```
# ('USA') is string not tuple  
t = ('USA')  
print(type(t))
```

```
<class 'str'>
```

```
# tuple with mixed values: str, int, bool, float  
t = ('USA', 2, True, 9.123)      # with parenthesis  
print(type(t))
```

```
<class 'tuple'>
```

```
# tuple in a tuple  
# tuple with mixed values: str, int, bool, float, tuple  
# (10,20,30) is a tuple in the tuple t.  
  
t = ('USA', 2, True, 9.123, (10,20,30))      # with parenthesis  
print(type(t))
```

```
<class 'tuple'>
```

## tuple() function

- The built-in `tuple()` function converts a string into a tuple, where each character of the string becomes an individual value in the tuple

```
t = tuple('Hello')  # convert string to tuple  
  
print(f'Type of t: {type(t)}')  
print(f't      : {t}')
```

[Skip to main content](#)

```
Type of t: <class 'tuple'>
t      : ('H', 'e', 'l', 'l', 'o')
```

- The built-in `tuple()` function converts a range into a tuple, encapsulating a sequence of numbers within it.

```
r = range(2,8)    # 2,3,4,5,6,7 are hidden in r
print(f'Type of r: {type(r)}')
print(f'r      : {r}')
```

```
Type of r: <class 'range'>
r      : range(2, 8)
```

```
t = tuple(r)    # convert range to tuple
print(f'Type of t: {type(t)}')
print(f't      : {t}')
```

```
Type of t: <class 'tuple'>
t      : (2, 3, 4, 5, 6, 7)
```

## Functions on tuples

The following functions can take a tuple as input and return:

- `len()`: the number of elements in a tuple.
- `max()`: the maximum value in a tuple.
  - For strings, dictionary order is used, and `max()` returns the last string in the dictionary order.
- `min()`: the minimum value in a tuple.
  - For strings, dictionary order is used, and `min()` returns the first string in the dictionary order.
- `sum()`: returns the sum of the elements (if they can be added) in a tuple.
  - It does not work with strings.
  - It works for booleans: True is 1, False is 0.

[Skip to main content](#)

## Examples

```
numbers = (7,3,1,9,6,4)

print(f'Length : {len(numbers)}')
print(f'Maximum: {max(numbers)}')
print(f'Minimum: {min(numbers)}')
print(f'Sum     : {sum(numbers)}')
```

Length : 6  
Maximum: 9  
Minimum: 1  
Sum : 30

```
letters = ('r', 't', 'n', 'a', 'd')

print(f'Length : {len(letters)}')
print(f'Maximum: {max(letters)}')      # dictionary order
print(f'Minimum: {min(letters)}')
```

Length : 5  
Maximum: t  
Minimum: a

```
numbers = (7,3,1,9,6,4,True)      # True is considered as 1

print(f'Length : {len(numbers)}')
print(f'Maximum: {max(numbers)}')
print(f'Minimum: {min(numbers)}')
print(f'Sum     : {sum(numbers)}')
```

Length : 7  
Maximum: 9  
Minimum: 1  
Sum : 31

```
t = (7,3,1,9,6,4,True, 'a')

print(f'Length : {len(t)}')      # only len() works for this tuple
```

[Skip to main content](#)

Length : 8

## Indexing and Slicing

- It is similar to strings.

```
t = ('USA', 2, True, 9.123, 'NY', 'NJ', 100, False)
```

### Examples

```
# first element  
print(t[0])
```

USA

```
# last element  
print(t[-1])
```

False

```
# index 3 element (fourth element)  
print(t[3])
```

9.123

```
# index=2,3,4  
print(t[2:5])
```

(True, 9.123, 'NY')

```
# index=-4,-3,-2  
print(t[-4:-1])
```

```
('NY', 'NJ', 100)
```

```
# slice starting from the index 3 element and all the way to the end  
print(t[3:])
```

```
(9.123, 'NY', 'NJ', 100, False)
```

## Remark

- There is a difference between the index -1 element and the slice [-1:].
- Both of them point to the last element of the tuple.
- The first one returns the last element, whereas the latter one returns a length-one tuple with the last element.

```
print(f'index -1 element: {t[-1]}, type: {type(t[-1])}')      # boolean  
print(f'slice [-1:]     : {t[-1:]}, type: {type(t[-1:])}')    # tuple
```

```
index -1 element: False, type: <class 'bool'>  
slice [-1:]     : (False,), type: <class 'tuple'>
```

## Remark

- A tuple in a super tuple is considered a single element of the super tuple.
- Its elements are not considered elements of the super tuple.

```
t = ('USA', 2, True, 9.123, (10,20,30))  
print(f'Length of t      : {len(t)}')           # (10,20,30) is a single element of t  
print(f'10 is in t      : {10 in t}')          # 10 is not an element of t  
print(f'(10,20,30) is in t: {(10,20,30) in t}') # (10,20,30) is an element of t
```

```
Length of t      : 5
10 is in t      : False
(10,20,30) is in t: True
```

**Remark** It is possible to access the elements of the sub tuple by using chain indexing.

```
t = ('USA', 2, True, 9.123, (10,20,30))
print(f't[-1]: {t[-1]}')           # t[-1] = (10,20,30) is a tuple
print(f't[-1][0]: {t[-1][0]}')     # indexing of t[-1] = (10,20,30)
print(f't[-1][1]: {t[-1][1]}')
print(f't[-1][2]: {t[-1][2]}')
```

```
t[-1]: (10, 20, 30)
t[-1][0]: 10
t[-1][1]: 20
t[-1][2]: 30
```

## Operators on Tuples

Operators behave similarly to strings.

- `+`: Concatenation
- `*`: Repetition (only integers are used)
- `in` and `not in` operators: check whether a value is an element in a tuple.
  - Returns a boolean value.

### Examples

```
numbers = (1,2,3,4)
letters = ('a','b','c','d')
```

```
# Concatenation returns a new tuple

print(f'numbers + letters = {numbers + letters}')
print(f'numbers          = {numbers}')          # no change
print(f'letters         = {letters}')          # no change
```

```
numbers + letters = (1, 2, 3, 4, 'a', 'b', 'c', 'd')
numbers           = (1, 2, 3, 4)
letters           = ('a', 'b', 'c', 'd')
```

```
# Repetition returns a new tuple
print(f'letters*3 = {letters*3}')
print(f'letters  = {letters}')      # no change
```

```
letters*3 = ('a', 'b', 'c', 'd', 'a', 'b', 'c', 'd', 'a', 'b', 'c', 'd')
letters  = ('a', 'b', 'c', 'd')
```

```
# Is 5 in numbers?
print(f' 5 is in numbers tuple : {5 in numbers}')
print(f' 5 is not in numbers tuple: {5 not in numbers}' )
```

```
5 is in numbers tuple : False
5 is not in numbers tuple: True
```

```
# Is 3 in numbers?
print(f' 3 is in numbers tuple : {3 in numbers}')
print(f' 3 is not in numbers tuple: {3 not in numbers}' )
```

```
3 is in numbers tuple : True
3 is not in numbers tuple: False
```

## Immutable

Similar to strings, tuples are immutable, which means they cannot be modified.

- For example, attempting to change the first element of a tuple will result in an error message.

```
# ERROR: try to change the first element, which has an index of 0.
t = (1,2,3,4)
```

[Skip to main content](#)

## Tuple Methods

- Except for the magic methods (those with underscores), there are only two methods for tuples. Y
- You can run `help(tuple)` for more details.

```
# methods of tuples  
print(dir(tuple))
```

```
['__add__', '__class__', '__class_getitem__', '__contains__', '__delattr__', '__dir__',
```

### count()

- It returns the number of occurrences of a given value in a tuple.

```
t = ('a', 'a', 'b', 'b', 'b', 'b', 'c')  
print(f'Number of a in t: {t.count("a")}')    # use " instead of ' for a  
print(f'Number of b in t: {t.count("b")}')  
print(f'Number of c in t: {t.count("c")}')  
print(f'Number of d in t: {t.count("d")}')
```

```
Number of a in t: 2  
Number of b in t: 4  
Number of c in t: 1  
Number of d in t: 0
```

### index()

- It returns the index of a given value in a tuple.
- If the value is not in the tuple, an error message is generated.
- In the case of repeated elements, it returns the smallest index

```
t = ('a', 'b', 'c', 'a')  
print(f'The index of a in t: {t.index("a")}')    # index of first a  
print(f'The index of b in t: {t.index("b")}')  
print(f'The index of c in t: {t.index("c")}')  
# print(f'The index of d in t: {t.index("d")}') ---> ERROR
```

[Skip to main content](#)

```
The index of a in t: 0  
The index of b in t: 1  
The index of c in t: 2
```

## Iterations and Tuples

We can use a *for* loop to access each element of a *tuple* and perform operations on each element. This can be done in two different ways:

1. Using the values in the tuple (Iterating through values).
2. Using indexes with the help of *len()* and *range()* functions (Iterating through indexes).
  - The largest index of a tuple is `length of the tuple - 1` since indexing starts from 0.
  - The indexes of a tuple are: 0, 1, 2, ..., length of the tuple – 1
  - By using the *range()* function, we can use `range(length of the tuple)`, which consists of all indexes.
    - Example: *range(5)* does not include the number 5; that's why we use the length of the tuple above.

The next two code snippets print the state names in the *states* tuple in two different ways.

```
states = ('Oklahoma', 'Texas', 'Florida', 'California')  
  
for state in states:  
    print(state)
```

```
Oklahoma  
Texas  
Florida  
California
```

```
states = ('Oklahoma', 'Texas', 'Florida', 'California')  
  
for i in range(len(states)):  
    print(states[i])
```

```
Oklahoma  
Texas  
Florida  
California
```

- While a *while* loop can also be used, the *for* loop is usually easier to work with when iterating through tuples.

```
states = ('Oklahoma', 'Texas', 'Florida', 'California')  
i = 0  
  
while i < len(states):  
    print(states[i])  
    i += 1
```

```
Oklahoma  
Texas  
Florida  
California
```

- Print the number of characters for each state.

```
states = ('Oklahoma', 'Texas', 'Florida', 'California')  
  
for state in states:  
    print(len(state))
```

```
8  
5  
7  
10
```

```
states = ('Oklahoma', 'Texas', 'Florida', 'California')  
  
for i in range(len(states)):  
    print(len(states[i]))      # states[i] is a state name
```

```
8  
5  
7  
10
```

## Examples

### Split even and odd numbers

Write a program that stores even numbers from the *numbers* tuple in a new tuple called *t\_even* and odd numbers in another tuple called *t\_odd*.

- Do not use lists.

```
numbers = (6,2,9,1,2,12,5,9,3,5,7,2,1,78,43,23,67,65,32,34,76,54)
```

#### Solution:

```
t_odd = ()  
t_even =()  
  
for i in numbers:  
    if i%2 == 0:  
        t_even += (i,)      # concatenation of tuples t_even and (i,)  
    else:  
        t_odd += (i,)  
  
print('Even Numbers:', t_even)  
print('Odd  Numbers:', t_odd)
```

```
Even Numbers: (6, 2, 2, 12, 2, 78, 32, 34, 76, 54)  
Odd  Numbers: (9, 1, 5, 9, 3, 5, 7, 1, 43, 23, 67, 65)
```

### Split data types

Write a program that stores the strings in the *mix\_tuple* tuple into a tuple called *t\_string*, integers into *t\_integer*, floats into *t\_float*, and booleans into *t\_boolean*.

[Skip to main content](#)

- The `mix_tuple` contains only strings, integers, floats, and boolean values.
- Do not use lists.

```
mix_tuple = (1, 3, 'NJ', False, 'OK', 5, 8, 'Hello', 9.8, True, 9, 87)
```

### Solution:

```
t_string, t_number, t_boolean = (), (), ()  
  
for i in mix_tuple:  
    if type(i) == str:  
        t_string += (i,)  
    elif type(i) == bool:  
        t_boolean += (i,)  
    else:  
        t_number += (i,)  
  
print('Strings :', t_string)  
print('Numbers :', t_number)  
print('Booleans:', t_boolean)
```

```
Strings : ('NJ', 'OK', 'Hello')  
Numbers : (1, 3, 5, 8, 9.8, 9, 87)  
Booleans: (False, True)
```

## Tuples Debugging

- Each of the following short code contains one or more bugs.
- Please identify and correct these bugs.
- Provide an explanation for your answer.

## Question

```
x = (1,2,3,4,5,6,7,8,9]
```

### Solution

The right square bracket `[ ]` must be a parenthesis `( )` so `x` can be a tuple.

## Question

```
x, y = 'NY', 'NJ'  
x*y
```

### Solution

The `*` operator cannot be used for two strings; it can only be used with either two numbers or one string and one integer.

## Question

```
x = ('A', 'B', 'C', 'D', 'E')  
x[2] = '-'
```

### Solution

`x` is a tuple, and tuples are immutable, so elements cannot be changed. `x[2]`, which is '`C`', cannot be changed as '`-`'.

## Question

```
x = ('A', 'B', 'C', 'D', 'E')  
x[-6]
```

### Solution

There is no element with index -6 because the negative indexes are: -1 for '`E`', -2 for '`D`', -3 for

[Skip to main content](#)

## Question

```
sequence = (1,2,3,4,5)
sequence.append(6)
```

### Solution

`sequence` is a tuple, so it is immutable. It cannot be modified, and new elements cannot be added to it. Therefore, tuples do not have an `append()` method.

## Question

```
sequence = (1,2,3,4,5)

for i in range(5):
    print(sequence[i+1])
```

### Solution

If  $i=4$ , `sequence[4+1]` refers to the element at the tuple index of 5, but the largest index is 4 (index of element 5). Therefore, `sequence[5]` does not exist.

## Question

```
sequence = ('a','b','c')
print(sequence.index('A'))
```

### Solution

The `index()` method raises an error since 'A' is not a member of the sequence tuple.

- Find the output of the following code.
- Please don't run the code before giving your answer.

## Question

```
x = ('a', 'b', 'c', 1, 2, 3, 'a')  
print(x.count('a'))  
print(x.count('c'))  
print(x.count('B'))
```

► Show code cell output

## Question

```
x = ('a', 'b', 'c', 1, 2, 3, 'a')  
print(x.index('a'))  
print(x.index('c'))
```

► Show code cell output

## Question

```
x = ('table', 'chair', 'desk', 'bookcase')  
for i in x:  
    print(x[-2])
```

► Show code cell output

## Question

```
x = ('table', 'chair', 'desk', 'bookcase')  
for i in x:  
    print(i[-2])
```

[Skip to main content](#)

## Question

```
x = ('A', 'B', 'C')

for i in range(len(x)):
    print(i)
```

► Show code cell output

## Question

```
x = ('A', 'B', 'C')

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

► Show code cell output

## Question

```
x = (12,5,3,99,123,10)

for i in x:
    if i>20:
        print(i)
```

► Show code cell output

## Question

```
x = (12,5,3,99,123,10)

for i in x:
    if i>20:
        if i<100:
            print(i)
```

► Show code cell output

[Skip to main content](#)

## Question

```
n = 0  
my_tuple = ('A', 'B', 'C', 'D', 'E', 'F')  
  
for i in my_tuple:  
    n += 1  
  
print(n)
```

► Show code cell output

## Question

```
total = 0  
  
for i in (10, 20, 30, 40):  
    total += i/5  
  
print(total)
```

► Show code cell output

## Question

```
total = 0  
  
for i in (10, 20, 30, 40):  
    total += i//8  
  
print(total)
```

► Show code cell output

## Question

[Skip to main content](#)

```
total = 0

for i in (10, 20, 30, 40):
    total += i/10
    total = i

print(total)
```

► Show code cell output

## Tuples Code

- Please solve the following questions using Python code.

### Question

Create a tuple with the following country names, including repetitions, and print the country with the largest length ('Netherlands') in two different ways.

1. Use an index to print 'Netherlands'.
2. Use a for loop to find the country with the greatest length."

- Germany
- Italy
- France
- Germany
- Netherlands
- Brazil
- Italy

### Solution-1

► Show code cell content

### Solution-2

► Show code cell content

[Skip to main content](#)

## Question

Write a program that prints the larger value in each pair in the following tuple in a single line.

- The output should be: 5, 10, 6, 8, 10, 11.

```
pairs = ((2,5), (10,2), (5,6), (8,2), (7,10), (11,3))
```

## Solution

▶ Show code cell content

## Question

Write a program that stores the larger value in each pair from the following tuple in another tuple.

- The output should be: (5, 10, 6, 8, 10, 11)

```
pairs = ((2,5), (10,2), (5,6), (8,2), (7,10), (11,3))
```

## Solution

▶ Show code cell content

## Question

Write a program that creates (name, grade) pairs by using the *names* and *grades* tuples, and stores these pairs in a tuple.

- The output should be : (('Lucas', 90), ('Henry', 75), ('Noah', 65), ('Cole', 100), ('Emma', 80), ('Camila', 70))

```
grades = (90, 75, 65, 100, 80, 70)
names = ('Lucas', 'Henry', 'Noah', 'Cole', 'Emma', 'Camila' )
```

## Solution

[Skip to main content](#)

▶ Show code cell content

## Question

Count the number of boolean values in the following tuple.

```
mytuple = (1, 2, 3, 4, True, False, 'USA', 'True', 'False', True, 8.75, False, 'NY')
```

## Solution

▶ Show code cell content

## Question

Find the sum of the numbers in the following list.

```
mytuple = ('A', 3, 'Florida', 8, 'B', 7, True, False, 'Hello', 12, 'NY')
```

## Solution

▶ Show code cell content

## Question

Use the split method to identify words in the provided text that meet the following criteria:

- Have a length of 4 or 5.
- Are not stop words (common words in English given in *stopwords* tuple below).
- Contain only letters (use the *isalpha()* method).
- Store the identified words in a tuple.

```
text = """ Imyep jgsqewt okbxsq seunh many rkx vmysz ndpoz may vxabckewro topfd tqkj ue
"""
```

[Skip to main content](#)

```
from sklearn.feature_extraction.text import ENGLISH_STOP_WORDS
stopwords = tuple(ENGLISH_STOP_WORDS)
print(f' First 5 stopwords: {stopwords[:5]}')
```

First 5 stopwords: ('itself', 'too', 'de', 'few', 'through')

## Solution

► Show code cell content

## Question

Given a tuple of 2-element tuples representing x and y coordinates of points, determine the 2-element tuple that is closest to the point (1, 2).

Distance formula:  $d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$ .

```
coordinates = ((-2,4),(5,-2),(6,-3),(0,5), (2,3), (-2,-3), (0,0))
```

## Solution

► Show code cell content

# Chp-8: Lists

- Learning Objectives
  - ..
  - ..

## Lists

Lists are similar to tuples as they are both ordered sequences of values of mixed types.

- The indexing, slicing, and functions on them are all similar to tuples

[Skip to main content](#)

- Lists are created using square brackets.
- Lists are mutable, allowing them to be modified. This is the main distinction.
  - Due to their mutability, lists have a large number of methods.
- The built-in `list()` function can be used to convert appropriate data types into a list.
- `[]` represents an empty list.

## Create Lists

```
# empty list
empty_list = []

print(type(empty_list))
```

```
<class 'list'>
```

```
# empty list with list()
empty_list = list()

print(type(empty_list))
```

```
<class 'list'>
```

```
# list with mixed values: str, int, bool, float

mixed_list = ['USA', 2, True, 9.123]
print(type(mixed_list))
```

```
<class 'list'>
```

```
# tuple and list in a list
# list with mixed values: str, int, bool, float, tuple, list
# (10,20,30) is a tuple and ['a','b'] is a list in the list mixed_list.

mixed_list = ['USA', 2, True, 9.123, (10,20,30), ['a','b']]
print(type(mixed_list))
```

[Skip to main content](#)

```
<class 'list'>
```

## list() function

- The built-in `list()` function converts a string into a list, where each character of the string becomes an individual value in the list.

```
char_list = list('Hello') # convert string to tuple  
  
print(f'Type of char_list: {type(char_list)}')  
print(f'char_list : {char_list}')
```

```
Type of char_list: <class 'list'>  
char_list : ['H', 'e', 'l', 'l', 'o']
```

- The built-in `list()` function converts a range into a list, encapsulating a sequence of numbers within it.

```
r = range(2,8) # 2,3,4,5,6,7 are hidden in r  
  
print(f'Type of r: {type(r)}')  
print(f'r : {r}')
```

```
Type of r: <class 'range'>  
r : range(2, 8)
```

```
num_char = list(r) # convert range to list  
  
print(f'Type of num_char: {type(num_char)}')  
print(f'num_char : {num_char}')
```

```
Type of num_char: <class 'list'>  
num_char : [2, 3, 4, 5, 6, 7]
```

- The built-in `list()` function converts a tuple into a list.

[Skip to main content](#)

```
t = (10,20,30)

sample_list = list(t)    # tuple ---> list

print(f'Type of sampleList: {type(sample_list)}')
print(f'sample_list      : {sample_list}')
```

```
Type of sampleList: <class 'list'>
sample_list      : [10, 20, 30]
```

- The built-in `tuple()` function converts a list into a tuple.

```
sample_list = [10,20,30]

t = tuple(sample_list)    # list ---> tuple

print(f'Type of t: {type(t)}')
print(f't      : {t}')
```

```
Type of t: <class 'tuple'>
t      : (10, 20, 30)
```

## Functions on lists

- `len()`, `max()`, `min()`, and `sum()` functions behave similarly for lists."

```
numbers = [7,3,1,9,6,4]

print(f'Length : {len(numbers)}')
print(f'Maximum: {max(numbers)}')
print(f'Minimum: {min(numbers)}')
print(f'Sum     : {sum(numbers)}')
```

```
Length : 6
Maximum: 9
Minimum: 1
Sum     : 30
```

[Skip to main content](#)

```
letters = ['r', 't', 'n', 'a', 'd']

print(f'Length : {len(letters)}')
print(f'Maximum: {max(letters)}')      # dictionary order
print(f'Minimum: {min(letters)}')
```

```
Length : 5
Maximum: t
Minimum: a
```

## Indexing and Slicing

- It is similar to strings and tuples.

```
mixed_list = ['USA', 2, True, 9.123, 'NY', 'NJ', 100, False]
```

### Examples

```
# first element
print(mixed_list[0])
```

```
USA
```

```
# last element
print(mixed_list[-1])
```

```
False
```

```
# index 3 element (fourth element)
print(mixed_list[3])
```

```
9.123
```

[Skip to main content](#)

```
# index=2,3,4  
print(mixed_list[2:5])
```

```
[True, 9.123, 'NY']
```

```
# index=-4,-3,-2  
print(mixed_list[-4:-1])
```

```
['NY', 'NJ', 100]
```

```
# slice starting from the index 3 element and all the way to the end  
print(mixed_list[3:])
```

```
[9.123, 'NY', 'NJ', 100, False]
```

## Remark

- There is a difference between the index -1 element and the slice [-1:].
- Both of them point to the last element of the list.
- The first one returns the last element, while the latter one returns a length-one list with the last element.

```
print(f'index -1 element: {mixed_list[-1]}, type: {type(mixed_list[-1])}')      # boolean  
print(f'slice [-1:]      : {mixed_list[-1:]}, type: {type(mixed_list[-1:])}')    # list
```

```
index -1 element: False, type: <class 'bool'>  
slice [-1:]      : [False], type: <class 'list'>
```

## Remark

- A tuple in a list is considered a single element of the list.
- Its elements are not considered elements of the list.

[Skip to main content](#)

```

mixed_list = ['USA', 2, True, 9.123, (10,20,30)]
print(f'Length of mixed_list      : {len(mixed_list)}')           # (10,20,30) is a tuple
print(f'10 is in mixed_list      : {10 in mixed_list}')          # 10 is not an element
print(f'(10,20,30) is in mixed_list: {(10,20,30) in mixed_list}') # (10,20,30) is an element

```

```

Length of mixed_list      : 5
10 is in mixed_list      : False
(10,20,30) is in mixed_list: True

```

## Remark

- It is possible to access the elements of a subtuple by using chain indexing.

```

mixed_list = ['USA', 2, True, 9.123, (10,20,30)]
print(f'mixed_list[-1]: {mixed_list[-1]}')                      # mixed_list[-1] = (10,20,30) is a tuple
print(f'mixed_list[-1][0]: {mixed_list[-1][0]}')                # indexing of t[-1] = (10,20,30)
print(f'mixed_list[-1][1]: {mixed_list[-1][1]}')
print(f'mixed_list[-1][2]: {mixed_list[-1][2]}')

```

```

mixed_list[-1]: (10, 20, 30)
mixed_list[-1][0]: 10
mixed_list[-1][1]: 20
mixed_list[-1][2]: 30

```

## Operators on Lists

The operators `+`, `*`, `in`, and `not in` behave similarly to strings and tuples.

### Examples

```

numbers = [1,2,3,4]
letters = ['a','b','c','d']

```

```

# Concatenation returns a new list

print(f'numbers + letters = {numbers + letters}')
print(f'numbers            = {numbers}')           # no change
print(f'letters           = {letters}')           # no change

```

[Skip to main content](#)

```
numbers + letters = [1, 2, 3, 4, 'a', 'b', 'c', 'd']
numbers           = [1, 2, 3, 4]
letters           = ['a', 'b', 'c', 'd']
```

```
# Repetition returns a new list
print(f'letters*3 = {letters*3}')
print(f'letters  = {letters}')      # no change
```

```
letters*3 = ['a', 'b', 'c', 'd', 'a', 'b', 'c', 'd', 'a', 'b', 'c', 'd']
letters  = ['a', 'b', 'c', 'd']
```

```
# Is 5 in numbers?
print(f' 5 is in numbers list : {5 in numbers}' )
print(f' 5 is not in numbers list: {5 not in numbers}' )
```

```
5 is in numbers list : False
5 is not in numbers list: True
```

```
# Is 3 in numbers?
print(f' 3 is in numbers list : {3 in numbers}' )
print(f' 3 is not in numbers list: {3 not in numbers}' )
```

```
3 is in numbers list : True
3 is not in numbers list: False
```

## Mutable

Unlike strings and tuples, lists are mutable, meaning they can be modified.

- Using list methods, new elements can be added, existing ones can be removed, and the order of elements can be changed.
- For example, you can change or delete the first element of a given list as follows:

[Skip to main content](#)

```
numbers = [1,2,3,4]
numbers[0] = 99 # first element is changed to 99
print(numbers)
```

```
[99, 2, 3, 4]
```

```
numbers = [1,2,3,4]
del numbers[0] # delete the first element
print(numbers)
```

```
[2, 3, 4]
```

## List Methods

Except for the magic methods (those with underscores), there are 11 methods for lists.

- You can execute `help(list)` for more details.

```
# methods of lists
# dir() returns a list

print(dir(list))
```

```
['__add__', '__class__', '__class_getitem__', '__contains__', '__delattr__', '__delitem__',
```

```
# non magic methods by using slicing
print(dir(list)[-11:])
```

```
['append', 'clear', 'copy', 'count', 'extend', 'index', 'insert', 'pop', 'remove', 'rev
```

`append()`

[Skip to main content](#)

```
numbers = [10,20,30]
print(f'numbers list before using append(): {numbers}')

# add 99 to numbers list
numbers.append(99)

print(f'numbers list after using append() : {numbers}')
```

```
numbers list before using append(): [10, 20, 30]
numbers list after using append() : [10, 20, 30, 99]
```

## clear()

It removes all elements from the list, turning it into an empty list.

```
numbers = [10,20,30]
print(f'numbers list before using clear(): {numbers}')

# remove all elemenets of numbers list
numbers.clear()

print(f'numbers list after using clear() : {numbers}')
```

```
numbers list before using clear(): [10, 20, 30]
numbers list after using clear() : []
```

## copy()

It returns a new list with the same elements as the original list.

```
numbers = [10,20,30]
print(f'numbers list      : {numbers}')

# copy of numbers list
numbers_copy = numbers.copy()

print(f'numbers_copy list: {numbers}')
```

[Skip to main content](#)

```
numbers list      : [10, 20, 30]
numbers_copy list: [10, 20, 30]
```

## count()

It returns the number of occurrences of a given value in a list.

```
numbers = [1,1,2,2,2,2,2,3,3,3,3,4]
print(f'Number of 1  in numbers: {numbers.count(1)}')
print(f'Number of 2  in numbers: {numbers.count(2)}')
print(f'Number of 3  in numbers: {numbers.count(3)}')
print(f'Number of 4  in numbers: {numbers.count(4)}')
print(f'Number of 5  in numbers: {numbers.count(5)}')
```

```
Number of 1  in numbers: 2
Number of 2  in numbers: 5
Number of 3  in numbers: 4
Number of 4  in numbers: 1
Number of 5  in numbers: 0
```

## extend()

It adds all elements of one list to another list using the form list1.extend(list2).

- All elements of list2 will be added to list1, and there will be no change to list2.

```
numbers = [10, 20, 30]
letters = ['a','b','c','d']
print(f'letters list before extending: {letters} ---- numbers list before extending: {numbers}

numbers.extend(letters)

print(f'letters list after extending : {letters} ---- numbers list after extending : {numbers}'")
```

```
letters list before extending: ['a', 'b', 'c', 'd'] ---- numbers list before extending
letters list after extending : ['a', 'b', 'c', 'd'] ---- numbers list after extending
```

It returns the index of a given value in a list.

- If the value is not in the list, an error message is generated.
- In the case of repeated elements, it returns the smallest index.

```
numbers = [10, 20, 30, 10]
print(f'The index of 10 in numbers: {numbers.index(10)}')    # index of first 10
print(f'The index of 20 in numbers: {numbers.index(20)}')
print(f'The index of 30 in numbers: {numbers.index(30)}')
# print(f'The index of 40 in numbers: {numbers.index(40)}') ----> ERROR
```

```
The index of 10 in numbers: 0
The index of 20 in numbers: 1
The index of 30 in numbers: 2
```

## insert()

It adds a new value to a list. Apart from the *append()* method the new value can be added to anywhere in the list.

- It is in the form of `insert(index, element)`
  - The element will be added to the index position.
- Example: `insert(3, 'USA')` → 'USA' will be the index of 3 element in the list.

It adds a new value to a list. Apart from the *append()* method, the new value can be added anywhere in the list using the form *insert(index, element)*.

- The element will be added to the specified index position.
- For example, `insert(3, 'USA')` will make 'USA' the element at index 3 in the list.

```
numbers = [10, 20, 30]
print(f'numbers list before using insert(): {numbers}')

# add 99 to numbers list as index of 2 element
numbers.insert(2, 99)    # index of 99 is 2

print(f'numbers list after using index() : {numbers}')
```

```
numbers list before using insert(): [10, 20, 30]
numbers list after using index() : [10, 20, 99, 30]
```

## pop()

It removes an element from the list using the index of the element.

- pop(i): removes the element at the i-th index.
- pop(): removes the last element.
- It also returns the removed element.

```
numbers = [10,20,30]
print(f'numbers list before using pop(): {numbers}')

# remove the last element
removed_element = numbers.pop()    # last element (30) is removed.

print(f'numbers list after using pop() : {numbers}')
print(f'removed element           : {removed_element}'')
```

```
numbers list before using pop(): [10, 20, 30]
numbers list after using pop() : [10, 20]
removed element               : 30
```

```
numbers = [10,20,30]
print(f'numbers list before using pop(1): {numbers}')

# remove the index of 1 element which is 20
numbers.pop(1)    # 20 is removed.

print(f'numbers list after using pop() : {numbers}'')
```

```
numbers list before using pop(1): [10, 20, 30]
numbers list after using pop() : [10, 30]
```

## remove()

- If there is more than one occurrence of the given element in the list, only the first one will be removed

```
numbers = [10,20,30]
print(f'numbers list before using remove(): {numbers}')

# remove 20
numbers.remove(20)

print(f'numbers list after using remove() : {numbers}')
```

```
numbers list before using remove(): [10, 20, 30]
numbers list after using remove() : [10, 30]
```

```
numbers = [10,20,30,20]
print(f'numbers list before using remove(): {numbers}')

# remove the first 20
numbers.remove(20)

print(f'numbers list after using remove() : {numbers}')
```

```
numbers list before using remove(): [10, 20, 30, 20]
numbers list after using remove() : [10, 30, 20]
```

## reverse()

It reverses the order of the elements in a list.

```
numbers = [10,20,30]
print(f'numbers list before using reverse(): {numbers}')

# reverse
numbers.reverse()

print(f'numbers list after using reverse() : {numbers}')
```

```
numbers list before using reverse(): [10, 20, 30]
numbers list after using reverse() : [30, 20, 10]
```

[Skip to main content](#)

## sort()

It sorts the elements of a list in ascending or descending order.

- If the values are strings, dictionary order will be used.
- The default ordering is ascending. To have a descending order, the parameter reverse is set to True.

```
numbers = [10,3,9,2,5,12,1,8]
print(f'numbers list before using sort(): {numbers}')

# ascending order
numbers.sort()

print(f'numbers list after using sort() : {numbers}')
```

```
numbers list before using sort(): [10, 3, 9, 2, 5, 12, 1, 8]
numbers list after using sort() : [1, 2, 3, 5, 8, 9, 10, 12]
```

```
numbers = [10,3,9,2,5,12,1,8]
print(f'numbers list before using sort(): {numbers}')

# descending order
numbers.sort(reverse=True)

print(f'numbers list after using sort() : {numbers}')
```

```
numbers list before using sort(): [10, 3, 9, 2, 5, 12, 1, 8]
numbers list after using sort() : [12, 10, 9, 8, 5, 3, 2, 1]
```

```
letters = ['y', 'c', 'z','t','d']
print(f'numbers list before using sort(): {letters}')

# dictionary order
letters.sort()

print(f'numbers list after using sort() : {letters}')
```

```
numbers list before using sort(): ['y', 'c', 'z', 't', 'd']
numbers list after using sort() : ['c', 'd', 't', 'y', 'z']
```

[Skip to main content](#)

```
letters = ['y', 'c', 'z','t','d']
print(f'numbers list before using sort(): {letters}')

# opposite dictionary order
letters.sort(reverse=True)

print(f'numbers list after using sort() : {letters}')
```

```
numbers list before using sort(): ['y', 'c', 'z', 't', 'd']
numbers list after using sort() : ['z', 'y', 't', 'd', 'c']
```

## Iterations and Lists

- It is similar to tuples.

```
# print state names in states list
states = ['Oklahoma', 'Texas', 'Florida', 'California']    # states is a list

for state in states:
    print(state)
```

```
Oklahoma
Texas
Florida
California
```

```
# print state names in states list
states = ['Oklahoma', 'Texas', 'Florida', 'California']    # states is a list

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

```
Oklahoma
Texas
Florida
California
```

```
# use a while loop
states = ['Oklahoma', 'Texas', 'Florida', 'California']      # states is a list
i = 0

while i < len(states):
    print(states[i])
    i += 1
```

Oklahoma  
Texas  
Florida  
California

```
# print the length of the state names
states = ['Oklahoma', 'Texas', 'Florida', 'California']      # states is a list

for state in states:
    print(len(state))
```

8  
5  
7  
10

```
# print the length of the state names
states = ['Oklahoma', 'Texas', 'Florida', 'California']      # states is a list

for i in range(len(states)):
    print(len(states[i]))          # states[i] is a state name
```

8  
5  
7  
10

## Lists and strings

Iterations and lists can be used together to analyze strings in a more efficient way.

[Skip to main content](#)

- Consider the following string:

```
text = """ Imyep jgsqewt okbxsq seunh many rkx vmysz ndpoz may vxabckewro topfd tqkj u
"""

```

- The `split()` method returns the words of the string in a list.
- It actually splits the string using the default '' (space) value of the `sep` parameter.

```
words = text.split()

print(f'Type of words      : {type(words)}')
print(f'First five elements: {words[:5]}')
print(f'Last   five elements: {words[-5:]}')
```

```
Type of words      : <class 'list'>
First five elements: ['Imyep', 'jgsqewt', 'okbxsq', 'seunh', 'many']
Last   five elements: ['486', 'sckpyhnf', 'mxa', 'qvceb.', 'Thus.']}
```

- The number of words (including the numbers) in the `text` is equal to the length of the `words` list.

```
words = text.split()

print(f'Number of words: {len(words)}')
```

```
Number of words: 1300
```

- Print the words in `text` with a length of 11 and that are not numbers.

```
for word in words:
    if len(word) == 11:
        if not word.isdigit():
            print(word)
```

```
xngcfmrosb.
dulhbmfkwt.
akcqxwshtj.
yweuqvndil.
otgxwczfjm.
```

- Print the word(s) in *text* that starts with 'r'!

```
for word in words:  
    if word.startswith('r'):      # you can also use word[0] == 'r'  
        print(word)
```

rkx  
rsn  
rbl  
rqwecv  
reg  
rgkfb  
rldqunvt  
rhnoiz  
rluzf  
rem.  
rfl  
rchui  
rbmwx  
rzni  
rkhvbj  
rfbti  
rgjyxd  
rfl  
rzj  
rcj  
rywae  
rfoscyclv  
rqmb  
rmixj  
re  
rtukhyl  
rqwgdyjx  
rvcwpuz  
rvbu  
rtpmu  
rpfomb  
rpyfh  
rmlp.  
rwu  
rqhg.  
roy  
rauh  
rpv  
rzodf.  
rlc  
rspdf  
rundkhfm  
rather  
ruiv

[Skip to main content](#)

- Print the word(s) in *text* with a length of 4 and end with 'h'

```
for word in words:
    if (len(word) == 4) & (word.endswith('h')):    # you can also use word[-1]=='h'
        print(word)
```

both  
bwmh  
jteh  
gsqh  
each  
rauh  
much  
xtfh  
such

- Construct a new list consisting of words with a length of 3.
- It is a common method to start with an empty list and fill it with the desired elements.

```
words3 = []

for word in words:
    if len(word) == 3:
        words3.append(word)          # len(word) == 3 ----> add it to words3 lists

print(f'The list of words with length 3: {words3}')
```

The list of words with length 3: ['rkx', 'may', 'bmt', 'nwr', 'rsn', 'kcq', 'ijt', 'ohs']

- Construct a new list consisting of words with a length of 3 and without any repetition.

```
words3_unique = []

for word in words:
    if (len(word) == 3) & (word not in words3_unique):      # if word is not already in
        words3_unique.append(word)

print(f'The list of unique words with length 3: {words3_unique}')
```

The list of unique words with length 3: ['rkx', 'may', 'bmt', 'nwr', 'rsn', 'kcq', 'ijt']

[Skip to main content](#)

# List Comprehension

It is a fast and concise way of creating lists in a single line using *for* loops and *if* statements.

- It is in the form of: `[expression for item in list]`
  - Here, the *expression* represents the elements of the list being constructed.
- An *if* statement can also be included in a list comprehension in the form of: `[expression for item in list if condition]`

## Example:

- The following code constructs a list with elements consisting of the squares of the integers between 1 and 10.

```
square_list = []
for i in range(1,11):
    square_list.append(i**2)
print(square_list)
```

```
[1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
```

- Let's use list comprehensions to construct a list with the same elements.

```
square_list = [i**2 for i in range(1,11)] # i**2 is the expression
print(square_list)
```

```
[1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
```

- If you compare the two codes above, using a list comprehension:
  - You do not need to initialize the `squares_list`.
  - Instead of three lines of code, you can construct the `squares_list` in a single line.

## Example:

- The following code constructs the list of squares of numbers between 1 and 100 whose first digit is

[Skip to main content](#)

- The integer  $i$  is converted to a string using the `str()` function.
- The first digit of  $i$  is the character at index 0 of the string representation, which is `str(i)[0]`.
- List comprehension makes the code much simpler.

```
square_list = []

for i in range(1,101):
    if str(i)[0] == '1':          # first digit of string i
        square_list.append(i**2)

print(square_list)
```

[1, 100, 121, 144, 169, 196, 225, 256, 289, 324, 361, 10000]

```
square_list = [i**2 for i in range(1,101) if str(i)[0]=='1']   # str(i)[0] is the first digit of string i

print(square_list)
```

[1, 100, 121, 144, 169, 196, 225, 256, 289, 324, 361, 10000]

## Example

- Construct a list consisting of player numbers in the form of `player_NUMBER` for the first 5 players.
- In the code, `player_` is a string, and  $i$  is an integer. To enable concatenation,  $i$  is converted to a string.

```
player_list = [ 'player_'+str(i) for i in range(1,6)]
print(player_list)
```

['player\_1', 'player\_2', 'player\_3', 'player\_4', 'player\_5']

## Lists Debugging

- Each of the following short code contains one or more bugs.

[Skip to main content](#)

- Provide an explanation for your answer.

## Question

```
x = ['A' 'B' 'C' 'D' 'E']
```

### Solution

Elements must be comma-separated.

## Question

```
x = ['A', 'B', 'C', 'D', 'E']  
x(2)
```

### Solution

Indexes must be inside square brackets. \*x(2) must be x[2].

## Question

```
x = ['A', 'B', 'C', 'D', 'E']  
x[len(x)]
```

### Solution

There is no element at the index  $\text{len}(x)$  since indexing starts from 0. The largest index is  $\text{len}(x)-1$ .

## Question

[Skip to main content](#)

```
mylist[0] [3]
```

### 🔔 Solution

- mylist[0] is the index of 0 element of mylist, which is ['NY', 'CA', 'FL'].
- mylist[0][3] is attempting to access the index 3 of ['NY', 'CA', 'FL'], but that element does not exist.

## Question

```
mylist = [['NY', 'CA', 'FL'], ('Amy', 'John', 'Ashley', 'Michael')]  
mylist[1] [5]
```

### 🔔 Solution

- mylist[1] is the index of 1 element of mylist which is ('Amy', 'John', 'Ashley', 'Michael').
- mylist[1][5] is the index of 5 element of ('Amy', 'John', 'Ashley', 'Michael') which does not exist.

## Lists Output

- Find the output of the following code.
- Please don't run the code before giving your answer.

## Question

```
for i in ['Utah', 'Texas', 'Florida']:  
    print(i[1])
```

▶ Show code cell output

## Question

[Skip to main content](#)

```
for i in ['Mike', 'Jack', 'Liz', 'Ted', 'John', 'Ashley']:
    print(f'Hello {i}')
```

► Show code cell output

## Question

```
for i in ['Mike', 'Jack', 'Liz', 'Ted', 'John', 'Ashley']:
    print(f'Hello {i}')
```

► Show code cell output

## Question

```
for i in ['Mike', 'Jack', 'Liz', 'Ted', 'John', 'Ashley', 'Bob']:
    if i[1] == 'o':
        print(i)
```

► Show code cell output

## Question

```
for i in ['Mike', 'Jack', 'Liz', 'Ted', 'John', 'Ashley', 'Bob', ['A', 'B']]:
    if len(i)<4:
        print(i)
```

► Show code cell output

## Question

```
for i in ['Mike', 'Jack', 'Liz']:
    print(f'{i.lower()} ---- {i.upper()}')
```

► Show code cell output

## Question

[Skip to main content](#)

```
n = 0
my_list = [3,1,6,-3,-5,7,8,9]
for i in my_list:
    if i >5:
        n += i
print(n)
```

► Show code cell output

## Question

```
p = 1
my_numbers = [2,-4,-3,5]
for i in my_numbers:
    if i <4:
        p *= i
print(p)
```

► Show code cell output

## Question

```
n = 0
my_list = [3,1,6,-3,-5,7,8,9]
for i in range(len(my_list)):
    if i >5:
        n += my_list[-i]
print(n)
```

► Show code cell output

## Question

```
my_list = [1,2,3,4,5]
for i in range(3):
    my_list[i] = 10*i
print(my_list)
```

► Show code cell output

[Skip to main content](#)

## Question

```
x = []
while True:
    x.append(2)
    if len(x) == 4:
        break
print(x)
```

► Show code cell output

## Question

```
mylist = [['NY', 'CA', 'FL'], ('Amy', 'John', 'Ashley', 'Michael')]
print(mylist[0][-2])
print(mylist[1][1])
print(mylist[1][:2])
```

► Show code cell output

## Lists Code

- Please solve the following questions using Python code.

## Question

Prompt the user for a name and store each character of the name as a lowercase letter in a list, avoiding the use of the `list()` method.

## Solution

[Skip to main content](#)

## Solution

```
name = input('Enter your name: ')
char_list = []
for i in name:
    char_list.append(i.lower())
print(f'Letters: {char_list}')
```

### Sample Output

Enter your name: Amelia

Leters: ['a', 'm', 'e', 'l', 'i', 'a']

## Question

Prompt the user for comma-separated numbers and store them in a list, where each number is treated as an element with a data type of float.

- Use only one input function and make use of the split() method for string parsing.

### Solution

## Solution

```
numbers = input('Enter numbers in a comma separated form: ')
number_list_str = numbers.split(',')
number_list_float = []

for i in number_list_str:
    number_list_float.append(float(i))

print(f'Numbers in list as float: {number_list_float}')
```

### Sample Output

Enter numbers in a comma separated form: 3,8,5,1,7

Numbers in list as float: [3.0, 8.0, 5.0, 1.0, 7.0]

[Skip to main content](#)

## Question

Use a for loop to identify numbers in the number\_list that are divisible by 5 and greater than 23, storing them in a new list named five\_list.

```
number_list = [58, 82, 9, 25, 11, 36, 43, 27, 95, 75]
```

## Solution

► Show code cell content

## Question

Use a for loop to reverse the elements of char\_list and store them in a new list named reverse\_list, without using any reverse method or indexing step.

```
char_list = ['k', 'p', 'f', 'l', 'j', 'l', 'u', 'g', 't', 'a']
```

## Solution-1

► Show code cell content

## Solution-2

► Show code cell content

## Solution-3

► Show code cell content

## Question

Use the split method to identify words in the given text that start with 'o' or 'O' excluding 'of,' 'or,' and 'on.'

- Store the identified words in a tuple named o\_tuple.

[Skip to main content](#)

```
text = """" Imyep jgsqewt okbxsq seunh many rkx vmysz ndpoz may vxabckewro topfd tqkj ue
```

## Solution

► Show code cell content

## Question

For the given number\_list, compute the cumulative sum and store it in a list named cum\_list.

Example: number\_list = [1,4,3,7,9,10]

Output:

- [1,1+4,1+4+3,1+4+3+7,1+4+3+7+9,1+4+3+7+9+10]
- [1,5,8,15,24,34]

```
number_list = [1, 4, 8, 1, 2, 4, 7, 5, 3, 3]
```

## Solution

► Show code cell content

## Question

Use a list comprehension to construct the following list.

- ['file\_1\_question', 'file\_2\_question',.....,'file\_5\_question']

## Solution

► Show code cell content

## Question

Construct a list using a list comprehension, which consists of the first letters of the given list

[Skip to main content](#)

```
name_list = ['Henry', 'Jack', 'Amanda', 'Ashley', 'Michael', 'Peter']
```

## Solution

▶ Show code cell content

## Question

Construct a string by concatenating the second letter of each string in the given list.

- Hint: Access and concatenate 'e', 'a', 'm', 's', 'i', 'e

```
name_list = ['Henry', 'Jack', 'Amanda', 'Ashley', 'Michael', 'Peter']
```

## Solution

▶ Show code cell content

## Question

Prompt the user for a name and store its lowercase letters in one list and uppercase letters in another, excluding any non-alphabetical characters.

## Solution

### Solution

```
name = input('Enter a name: ')

lower_list, upper_list = [], []
for i in name:
    if i.isalpha():
        if i == i.lower():
            lower_list.append(i)
        else:
            upper_list.append(i)

print(f'Lower Case Letters: {lower_list}')
```



[Skip to main content](#)

## Sample Output

Enter a name: miCHaEl

Lower Case Letters: ['m', 'i', 'a', 'l']

Upper Case Letters: ['C', 'H', 'E']

## Question

Generate a list of numbers from 1 to 9, then use the shuffle method from the random library to alter the order of these numbers.

- Display the resulting list in a table format.
- Example: shuffled\_list = [7, 8, 3, 1, 2, 4, 5, 9, 6]
  - Output:

\_\_\_\_\_  
|7|8|3|  
\_\_\_\_\_

|1|2|4|  
\_\_\_\_\_

|5|9|6|  
\_\_\_\_\_

## Solution-1

► Show code cell content

## Solution-2

► Show code cell content

## Question

Create a list of length 10 with values randomly chosen from 'P' or 'N', where 'P' represents positive and 'N' represents negative test results.

- Use either the random module or numpy.random.

[Skip to main content](#)

- Example
  - old\_list = ['M', 'F', 'M', 'M', 'F', 'M', 'F', 'M', 'M', 'F']
  - new\_list = [1, 0, 1, 1, 0, 1, 0, 1, 1, 0]

## Solution

▶ Show code cell content

## Question

For the given list, identify the maximum and minimum elements. Then, subtract the minimum value from each element in the list and divide the result by the difference between the maximum and minimum elements.

- Example: given list is [3, 8, 2, 9, 4, 12, 5]
- maximum value = 12, minimum value = 2
- subtract 2 from each element: [1, 6, 0, 7, 2, 10, 3]
- divide each element by max - min =  $12 - 2 = 10$ 
  - output: [0.1, 0.6, 0, 0.7, 0.2, 1, 0.3]

```
number_list = [3, 8, 2, 9, 4, 12, 5]
```

## Solution-1

▶ Show code cell content

## Solution-2

▶ Show code cell content

## Question

For the given list, calculate the mean and standard deviation.

- Subtract the mean from each element in the list and divide the result by the standard deviation.

[Skip to main content](#)

- You may use functions imported from either the numpy or statistics libraries for mean and standard deviation calculations.

```
number_list = [3, 8, 2, 9, 4, 12, 5]
```

## Solution-1

▶ Show code cell content

## Solution-2

▶ Show code cell content

## Question

Write a program that prompts the user to input a letter.

- Print the number of occurrences of this character in the provided text using a for loop.
- Additionally, store the index of each occurrence of this letter in a list.

```
text = """" Imyep jgsqewt okbxsq seunh many rkx vmysz ndpoz may vxabckewro topfd tqkj ue
```

### Solution

```
char = input('Please enter a character: ')
count = 0
index_list = []

for i in text:
    if i == char:
        count +=1
    if len(index_list) == 0:
        index_list.append(text.find(char))
    else:
        index_list.append(text.find(char, index_list[-1]+1))

print(f'Number of {char} is {count}.')
print(f'First 5 indexes: {index_list[:5]}')
```

[Skip to main content](#)

## Sample Output

Please enter a character: k

Number of k is 174.

First 5 indexes: [16, 34, 58, 72, 135]

## Question

Generate a list containing randomly generated five-letter strings, with a length of 10.

- Use the random.choice() function for the construction of the list.

## Solution

▶ Show code cell content

# Chp-9: Functions

- Learning Objectives
  - ..
  - ..

## Motivation

We have seen the following code related to the grading scale in the Conditionals chapter.

- It displays the corresponding letter grades according to the following chart.

Letter Grade	Grade Range
A	80 - 100
B	60 - 79
C	40 - 59
D	20 - 39
F	0 - 19

```
grade = 90

if 80 <= grade <= 100:
    print('Your letter grade is A')
elif 60 <= grade :
    print('Your letter grade is B')
elif 40 <= grade :
    print('Your letter grade is C')
elif 20 <= grade :
    print('Your letter grade is D')
elif 0 <= grade :
    print('Your letter grade is F')
else:
    print(f'{grade} is not a percent grade')
```

Your letter grade is A

If you need the letter grades of more than one students in your program then you need to copy and paste this code again and again with different grade values in your program.

- This will make your program very lengthy and hard to read.
- Instead of this you can use a *function* which is similar to the functions in mathematics and takes.
  - `grade` will be the input of the function and it will print the letter grade.
- The function version as follows and displays letter grades for three students.

If you need the letter grades for more than one student in your program, copying and pasting this code multiple times with different grade values will make your program very lengthy and hard to read.

- Instead you can use a function similar to functions in mathematics which takes grade as input and

[Skip to main content](#)

- The function version is as follows and displays letter grades for three students.

```
# this is the function named letter_grade
def letter_grade(grade):
    if 80 <= grade <= 100:
        print('Your letter grade is A')
    elif 60 <= grade :
        print('Your letter grade is B')
    elif 40 <= grade :
        print('Your letter grade is C')
    elif 20 <= grade :
        print('Your letter grade is D')
    elif 0 <= grade :
        print('Your letter grade is F')
    else:
        print(f'{grade} is not a percent grade')

# call function
letter_grade(80)
letter_grade(50)
letter_grade(30)
```

Your letter grade is A  
 Your letter grade is C  
 Your letter grade is D

- As you can see above, you just need to call the function by its name with the input value.
- This makes the code short and easy to read.

## Functions

Functions are used to avoid repetitions in a program by constructing reusable code.

- By using functions, a long code can be split into multiple functions like Lego bricks.
- In this way, it becomes easy to read and understand the code.
- A function can be called in a program with its name and parameters, and functions can include print statements.
- The structure of a function is as follows:

```
def function_name(parameters):
```

[Skip to main content](#)

```
def function_name(parameters):
    : 
        BLOCK_CODE
    return return_value
```

In the structure above:

- `def` is a keyword that initiates the construction of the function.
- `function_name` is the name of the function used to call it.
  - It is a good practice to choose meaningful names for functions to remember their purpose.
- `parameters` are the comma-separated inputs of the function.
  - A function can have no parameters, one, or more parameters.
- `:` comes right after the parameters, indicating that the following lines will be part of the function's code block.
- `BLOCK_CODE` is a group of code with the same indentation level that will be executed with the given parameter values.
- `return` is a keyword that terminates the function.
- `return_value` is the output of the function.
  - Some functions may not have a return statement.

### Example: Square Function

The following function is named  $f$ .

- Its parameter is  $x$  (a number).
- It calculates the square of  $x$ .
- It returns the square as its output.

```
def f(x):
    square = x**2
    return square
```

```
# call the function for x=3
print(f(3))
```

```
# call the function for x=5
print(f(5))
```

25

### Example: Area of a Rectangle

- The following function is named `area_rect`.
- It has two parameters: `width` and `height`.
- It calculates the area using the formula  $Area = Width \times Height$ .
- It returns the area as its output.

```
def area_rect(width, length):
    area = width*length
    return area
```

```
# call the function for width=5, height=10
print(area_rect(5, 10))
```

50

```
# call the function for width=8, height=9
print(area_rect(8, 9))
```

72

### Example: Area and Perimeter of a Circle

- The following function is named `circle_area_perimeter`.
- It has one parameter: `radius`.
- It calculates the area and perimeter of a circle with radius  $r$  using the formulas:  $area = \pi r^2$  and  $perimeter = 2\pi r$ .
- The calculated area and perimeter are rounded to the nearest hundredths.

[Skip to main content](#)

```
import math

def circle_area_perimeter(radius):
    area = math.pi*radius**2
    perimeter = 2*math.pi*radius
    area_round = round(area, 2)
    perimeter_round = round(perimeter, 2)
    return (area_round, perimeter_round)
```

```
# call the function for radius=5
print(circle_area_perimeter(5))
```

(78.54, 31.42)

```
# call the function for radius=10
print(circle_area_perimeter(10))
```

(314.16, 62.83)

## Example: Fahrenheit to Celcius Converter

- The following function is named conv\_f\_c.
- It has one parameter: fahrenheit.
- It calculates the equivalent Celsius value using the conversion formula:  $celsius = \frac{(fahrenheit - 32)}{1.8}$ .
- The Celsius value is rounded to the nearest hundredths.
- The function returns the Celsius value

```
def conv_f_c(fahrenheit):
    celcius = (fahrenheit-32)/1.8
    celcius_round = round(celcius, 2)
    return (celcius_round)
```

```
# call the function for fahrenheit=100
print(conv_f_c(100))
```

[Skip to main content](#)

```
# call the function for fahrenheit=20
print(conv_f_c(20))
```

-6.67

## No return statement

- It is possible to have functions with no return statement.
- Such functions usually include print statements.
- The following function takes a name as its input and then prompts for the age.

```
def age(name):
    print(f'How old are you {name}?')
```

```
age('Arthur')
```

How old are you Arthur?

- If you run the following code:
  - The print statement will be executed.
  - Since there is no return statement, no value will be returned, and x will be of type *NoneType*.

```
x = age('Arthur')
```

How old are you Arthur?

```
print(x)
```

None

```
print(type(x))
```

```
<class 'NoneType'>
```

## No parameters

- It is possible for a function to have no parameters.

```
def welcome():
    greeting = '''Good morning everyone,
I hope you are all doing well. It's a great pleasure for me to weilcome all of you to t
We have a very busy schedule today. Let's start working on each subject one by one. Tha
    return greeting
```

```
print(welcome())
```

```
Good morning everyone,
I hope you are all doing well. It's a great pleasure for me to weilcome all of you to t
We have a very busy schedule today. Let's start working on each subject one by one. Tha
```

## Default parameter values

If no value is given to the parameter, the default value will be used.

- Default values are typically assigned to parameters that are not frequently used or have a common default value.
- Non-default parameters should precede default parameters.

```
# course parameter has a default value
def student_report(name, grade, course='Math'):
    print(f'{course} grade of {name} is {grade}.')
```

```
# call student_report of Michael for CS  
student_report('Michael', 87, 'CS')
```

CS grade of Michael is 87.

```
# course value is not given so default value='Math' is used  
student_report('Michael', 87)
```

Math grade of Michael is 87.

```
# ERROR: Default parameter course cannot come before non-default parameter grade  
def student_report(name, course='Math', grade):  
    print(f'{course} grade of {name} is {grade}.')
```

## Local and Global Variables

- *Local variables* are defined inside a function and can only be accessed within that function.
- *Global variables* are defined outside a function and can be accessed inside a function, but they cannot be modified within the function.
  - When a global variable is called, a new local variable is used.

```
a = 4      # global variable  
  
def f(x):  
    b = 5    # local variable  
    result = a+b+x  
    return result
```

```
# call function for x=10  
print(f(10))
```

```
# you can acces 'a' outside the function: global variable  
print(a)
```

4

```
# ERROR: 'b' is not defined outside the function: local variable  
print(b)
```

```
# ERROR: 'result' is not defined outside the function: local variable  
print(result)
```

```
a = 4      # global variable  
  
def f(x):  
    a = 100      # change the value of a  
    b = 5  
    result = a+b+x  
    return result
```

```
# call function for x=10  
print(f(10))  # a=100 is used
```

115

```
# The value of 'a' outside the function has not been changed  
print(a)
```

4

## Examples

### Calculator

[Skip to main content](#)

- Operations are given as strings in the form of `'+', '*', '/', '-'`
- By using the given numbers and operation find `number1 operation number2`
- If the operation is division second parameter (denominator) can not be zero and display a warning message.
- If the operation is not one of the four operations given above display a warning message.

```
def calculator(number1, number2, operation):
    if operation == '+':
        return number1 + number2
    elif operation == '-':
        return number1 - number2
    elif operation == '*':
        return number1 * number2
    elif operation == '/':
        if number2 == 0:
            print('Warning: zero division')
        else:
            return number1/number2
    else:
        print(f'{operation} is not an available operation.')
```

```
# addition
print(calculator(7,2,'+'))
```

9

```
# subtraction
print(calculator(7,2,'-'))
```

5

```
# multiplication
print(calculator(7,2,'*'))
```

14

```
# division  
print(calculator(7,2,'/'))
```

3.5

```
# division by zero  
calculator(7,0,'/')
```

Warning: zero division

```
# inappropriate operation  
calculator(7,3,'%')
```

% is not an available operation.

## Functions Debugging

- Each of the following short code contains one or more bugs.
- Please identify and correct these bugs.
- Provide an explanation for your answer.

## Question

```
def f(x)  
    return x**2
```

### Solution



The missing colon in the first line should be added.

[Skip to main content](#)

## Question

```
def (x,y,z):  
    return x+y+z
```

### Solution

The name of the function is missing.

## Question

```
def f(x,y):  
    return x+y  
  
print(f(1,2,3))
```

### Solution

The function *f* has two parameters, but in the last line, three parameters are given.

## Question

```
def func(a,b,c):  
    x = a**2+b*4-c  
    return x  
  
func(1,5)
```

### Solution

The function `func` has three parameters (a, b, c). In the last line, only two inputs are provided for the `func()` function.

## Question

[Skip to main content](#)

```
def my_func(x):  
    return x+6  
  
my_func('3')
```

### Solution

The value '3' is a string, and inside the function, an attempt is made to add 6 to the string '3', resulting in an error.

## Question

```
def my_func(x):  
    return str(x)+6  
  
my_func(3)
```

### Solution

`str(x)` represents a string, while 6 is an integer. They cannot be added or concatenated directly.

## Question

```
def f(x):  
    y = 5  
    return x+y  
  
print(f(10))  
print(y)
```

### Solution

`y` is a local variable, and it cannot be accessed outside the function, as attempted in the last line.

[Skip to main content](#)

## Question

```
def f(y=0, x):  
    return x+y
```

### Solution

The non-default parameters must be placed before default parameters. The correct form is `f(x, y=0)`.

## Functions Output

- Find the output of the following code.
- Please don't run the code before giving your answer.

## Question

```
def f(x):  
    return 10*x
```

### Solution

No output.

## Question

```
def f(x):  
    return 10*x  
  
print(f(5))
```

► Show code cell output

[Skip to main content](#)

```
def f(x, y=5):  
    return x+y  
  
print(f(2, 4))  
print(f(10))
```

► Show code cell output

## Question

```
def func_letter():  
    print('A')  
    print('B')  
  
def func_word():  
    print('X')  
    func_letter()  
  
func_word()
```

► Show code cell output

## Question

```
def my_func(x):  
    print('A')  
    print('B')  
    return x**2  
  
print(my_func(3))
```

► Show code cell output

## Question

```
def func(a,b,c=3):  
    x = a**2+b**4-c  
    return x  
  
func(1,5)
```

[Skip to main content](#)

## Question

```
def my_func(x, y):  
    print(x//3)  
    y += 2  
    print(x+y)  
  
my_func(13,7)
```

► Show code cell output

## Question

```
def my_func(x, y, z):  
    if x+y > 10:  
        total = x+y  
    else:  
        total = y+z  
    return total  
  
print(my_func(2,7,4))
```

► Show code cell output

## Question

```
c = 3  
def f(x,y):  
    z = 6  
    return x+y+z+c  
  
print(f(1,2))  
c = 5  
print(f(1,2))
```

► Show code cell output

## Question

[Skip to main content](#)

```
def f(x, y):  
    print('x:', x)  
    return x+y  
  
print(f(1,2))
```

► Show code cell output

## Question

```
def f(x=1, y=6):  
    return x+y  
  
f()
```

► Show code cell output

## Functions Code

- Please solve the following questions using Python code.

## Question

Write a function that accepts an integer as its parameter and prints an  $n \times n$  square with '\*' characters.

## Solution

► Show code cell content

```
# example  
square_star(5)
```

```
* * * * *  
* * * * *  
* * * * *  
* * * * *  
* * * * *
```

[Skip to main content](#)

## Question

Write a function that takes a string as its parameter and returns the vowels ('a, e, i, o, u') in the given string without repetition as a list.

- If there are no vowels in the string, return the string 'No vowels' .

## Solution

▶ Show code cell content

```
# example
print(vowel_func('abbbecido'))
```

```
['a', 'e', 'i', 'o']
```

```
# example
print(vowel_func('bcd'))
```

No vowels!

## Question

Write a function that takes two integers as parameters and returns the powers of the first integer (base) from 0 to the second integer number.

- Example: parameters are 3, 5
  - Output:  $[3^0, 3^1, 3^2, 3^3, 3^4, 3^5]$

## Solution-1

▶ Show code cell content

```
# example
print(power_func(3,5))
```

[Skip to main content](#)

```
[1, 3, 9, 27, 81, 243]
```

## Solution-2

▶ Show code cell content

```
# example  
print(f(3,5))
```

```
[1, 3, 9, 27, 81, 243]
```

## Question

Write a function with parameters *start* and *end*, where it returns a list of all even numbers between start and end, inclusive.

- If start is greater than end, the output should be in descending order; if start is less than end, the output should be in ascending order.
- Example:
  - start=11, end=23 → Output: [12, 14, 16, 18, 20, 22]
  - start=20, end= 5 → Output: [20, 18, 16, 14, 12, 10, 8, 6]

## Solution-1

▶ Show code cell content

```
# example  
print(even_numbers(45, 23))
```

```
[44, 42, 40, 38, 36, 34, 32, 30, 28, 26, 24]
```

## Solution-2

▶ Show code cell content

[Skip to main content](#)

```
# example  
print(even_num(20,5))
```

```
[20, 18, 16, 14, 12, 10, 8, 6]
```

## Question

Write a function that takes two tuples as parameters and returns a tuple consisting of elements present in the first tuple but not in the second tuple.

- Example:
  - tuple1 = [1, 5, 9, 12, 6], tuple2 = [2, 7, 6, 9, 20, 23, 29] —> Output: [1, 5, 12]
  - tuple1 = [1,3], tuple2 = [2, 7] —> Output: [1, 3]

## Solution-1

► Show code cell content

```
# example  
print(diff_tuples( (1,2,3,4), (3,4,5,6) ))
```

```
[1, 2]
```

## Solution-2

► Show code cell content

```
# example  
print(diff_tuples( (1,2,3,4), (3,4,5,6) ))
```

```
(1, 2)
```

## Question

[Skip to main content](#)

Write a function that takes a date as a string in the format 'month/day/year', where the month is represented by a number.

- The function should return the date in the format 'year/month/day'.
- Example:
  - Input : '09/25/1987'
  - Output: '1987/09/25'

## Solution

▶ Show code cell content

```
# example
print(date_conv('09/25/1987'))
```

1987/09/25

## Question

Write a function that takes a list as its parameter and returns the standard deviation of its elements.

The standard deviation (std) for a sequence is calculated using the formula:  $std = \sqrt{\frac{\sum_{i=1}^n (x_i - \bar{x})^2}{n}}$

where  $\bar{x}$  is the mean and  $\sum$  denotes the sum. Here's a hint for the implementation:

1. Find the mean.
  2. Subtract mean from each element.
  3. Square the differences.
  4. Sum the squared differences.
  5. Divide by n
  6. Take square root
- Find the std of the list: [5,8,2,5,6,1,1,3]
  - Compare the result obtained from your custom standard deviation function with the standard

[Skip to main content](#)

## Solution

▶ Show code cell content

```
# example
mylist = [5,8,2,5,6,1,1,3]

print(f'Custom Function: {std_func(mylist)}')
print(f'Numpy       : {np.std(mylist)}')
print(f'Statistics  : {statistics.pstdev(mylist)}')
```

```
Custom Function: 2.368411915187052
Numpy       : 2.368411915187052
Statistics  : 2.368411915187052
```

## Chp-10: Sets

- Learning Objectives
  - ..
  - ..

Sets are an unordered collection of values.

- Since there is no order there is no indexing for sets.
- It can contain a mixed type of elements.
- Curly brackets `{}` are used to create sets.
  - **Warning:** `{}` is NOT an empty set. It is an empty *dictionary* that will be covered in later chapters.
- Sets are mutable. so they can be modified like lists.
- A tuple can be an element of a set.
- A list cannot be an element of a set.
- You can use the `set()` function to convert strings, tuples, and lists into a set.
  - Only unique values are stored in sets (no repetition).
  - **Warning:** You will lose the order of the elements when you use the conversion.
  - Example:

[Skip to main content](#)

- ```
my_list = [1,4,7,7,8]
my_set = set(my_list)
print(myset)
```

- Output: 1,4,7,8

## Create Sets

```
# empty set
empty_set = set()

print(f'Empty set : {empty_set}')
print(f'Type of empty_set: {type(empty_set)}')
```

```
Empty set : set()
Type of empty_set: <class 'set'>
```

```
# set with mixed values: str, int, bool, float
s = {'USA', 2, True, 9.123}
print(type(s))
```

```
<class 'set'>
```

```
# set removes the repetitions
s = {'USA', 2, True, 9.123, 'USA', 'USA', 'USA', 'USA'}
print(s) # only one 'USA' will be in the set s
```

```
{True, 'USA', 2, 9.123}
```

```
# tuple and set in a set
# set with mixed values: str, int, bool, float, tuple, set
# (10,20,30) is a tuple and ['a','b'] is a list in the list mixed_list.

s = ['USA', 2, True, 9.123, (10,20,30), {'a','b'}]
print(type(s))
```

```
<class 'list'>
```

```
# a list can not be an element of a set

s = {'USA', 2, True, 9.123, (10,20,30), ['a','b']}           # ERROR
```

## set() function

- The built-in `set()` function converts strings, tuples, and lists to a set, removing any duplicates and retaining only unique elements.

```
char_set = set('Hello') # convert string to set

print(f'Type of char_set: {type(char_set)}')
print(f'char_set      : {char_set}')
```

```
Type of char_set: <class 'set'>
char_set      : {'e', 'l', 'o', 'H'}
```

```
t = (1,2,3,4,4,4)
char_set = set(t) # convert tuple to set

print(f'Type of char_set: {type(char_set)}')
print(f'char_set      : {char_set}')
```

```
Type of char_set: <class 'set'>
char_set      : {1, 2, 3, 4}
```

```
number_list = [1,2,3,4,4,4]
char_set = set(number_list) # convert list to set

print(f'Type of char_set: {type(char_set)}')
print(f'char_set      : {char_set}')
```

```
Type of char_set: <class 'set'>
char_set      : {1, 2, 3, 4}
```

**Remark:** By using `tuple()` and `list()` functions, sets can be converted to tuples and lists, respectively.

- It's important to note that the conversion doesn't preserve any specific order, as sets are inherently unordered collections.

```
s = {1,2,3,4}    # set

s_tuple = tuple(s)      # set ---> tuple

print(f'Type of s_tuple: {type(s_tuple)}')
print(f's_tuple       : {s_tuple}')
```

```
Type of s_tuple: <class 'tuple'>
s_tuple      : (1, 2, 3, 4)
```

```
s = {1,2,3,4}    # set

s_list = list(s)      # set ---> list

print(f'Type of s_list: {type(s_list)}')
print(f's_list       : {s_list}')
```

```
Type of s_list: <class 'list'>
s_list      : [1, 2, 3, 4]
```

## Functions on sets

- `len()`, `max()`, `min()`, and `sum()` functions can be applied to sets, similar to other data types like

[Skip to main content](#)

```
numbers = {7,3,1,9,6,4}

print(f'Length : {len(numbers)}')
print(f'Maximum: {max(numbers)}')
print(f'Minimum: {min(numbers)}')
print(f'Sum     : {sum(numbers)}')
```

```
Length : 6
Maximum: 9
Minimum: 1
Sum    : 30
```

```
letters = {'r', 't', 'n', 'a', 'd'}

print(f'Length : {len(letters)}')
print(f'Maximum: {max(letters)}')      # dictionary order
print(f'Minimum: {min(letters)}')
```

```
Length : 5
Maximum: t
Minimum: a
```

## Set Operations

The available operators for sets in Python include `&` (intersection), `|` (union), `-` (difference), `^` (symmetric difference), `in`, and `not in`.

### Intersection

The `&` (ampersand) operator returns a new set consisting of the common elements of the two sets.

```
s1 = {1,2,3,4,5}
s2 = {3,4,5,6,7}

print(f'Intersection of s1 and s2: {s1&s2}')
```

```
Intersection of s1 and s2: {3, 4, 5}
```

[Skip to main content](#)

- The *intersection()* method of sets can also be used to find the common elements between two sets.

```
s1 = {1,2,3,4,5}
s2 = {3,4,5,6,7}

print(f's1 intersection s2 : {s1.intersection(s2)}')
print(f'No change on s1    : {s1}')
```

```
s1 intersection s2 : {3, 4, 5}
No change on s1    : {1, 2, 3, 4, 5}
```

## Union

The `|` (pipe) operator returns a new set consisting of the combined elements of the two sets.

```
s1 = {1,2,3,4,5}
s2 = {3,4,5,6,7}

print(f'Union of s1 and s2: {s1|s2}')
```

```
Union of s1 and s2: {1, 2, 3, 4, 5, 6, 7}
```

- The *union()* method of sets can also be used to combine elements from two sets into a new set.

```
s1 = {1,2,3,4,5}
s2 = {3,4,5,6,7}

print(f's1 union s2; {s1.union(s2)}')
print(f's1           : {s1}')          # No change on s1
```

```
s1 union s2; {1, 2, 3, 4, 5, 6, 7}
s1           : {1, 2, 3, 4, 5}
```

## Difference

[Skip to main content](#)

The `-` (dash) operator returns a new set consisting of the elements in the first set but not in the second set.

- `s1 - s2`: elements in `s1` but not in `s2`.
- `s2 - s1`: elements in `s2` but not in `s1`.

```
s1 = {1,2,3,4,5}
s2 = {3,4,5,6,7}

print(f's1 - s2: {s1-s2}')
print(f's2 - s1: {s2-s1}')
```

```
s1 - s2: {1, 2}
s2 - s1: {6, 7}
```

- The `difference()` method of sets can also be used to obtain a new set consisting of the elements in the first set but not in the second set.

```
s1 = {1,2,3,4,5}
s2 = {3,4,5,6,7}

print(f's1 - s2: {s1.difference(s2)}')
print(f's2 - s1: {s2.difference(s1)}')
```

```
s1 - s2: {1, 2}
s2 - s1: {6, 7}
```

## Symmetric Difference

The `^` (caret) operator returns a new set consisting of elements in either one of the two sets but not both.

- `s1 ^ s2` and `s2 ^ s1` are same.

```
s1 = {1,2,3,4,5}
s2 = {3,4,5,6,7}

print(f's1 ^ s2: {s1^s2}')
```

[Skip to main content](#)

```
s1 ^ s2: {1, 2, 6, 7}
s2 ^ s1: {1, 2, 6, 7}
```

- The `symmetric_difference()` method of sets can also be used to obtain a new set consisting of the elements in either one of the two sets but not both.

```
s1 = {1,2,3,4,5}
s2 = {3,4,5,6,7}

print(f's1 - s2: {s1.symmetric_difference(s2)}')
print(f's2 - s1: {s2.symmetric_difference(s1)}')
```

```
s1 - s2: {1, 2, 6, 7}
s2 - s1: {1, 2, 6, 7}
```

## in & not in

`in` checks if a value is part of a set, while `not in` verifies if a value is absent from a set.

- Both operations yield a Boolean result, either True or False.

```
s1 = {1,2,3,4,5}

print(f' 5 is in s1    : {5  in s1}' )
print(f' 5 is not in s1: {5 not in s1}' )
```

```
5 is in s1    : True
5 is not in s1: False
```

```
s1 = {1,2,3,4,5}

print(f' 9 is in s1    : {9  in s1}' )
print(f' 9 is not in s1: {9 not in s1}' )
```

```
9 is in s1    : False
9 is not in s1: True
```

# Mutable

Unlike strings and tuples, and similar to lists, sets are mutable, allowing them to be modified.

- The set methods enable the addition of new elements and the removal of existing ones.
- Set elements themselves cannot be changed, but new items can be added or existing ones removed.

## Set Methods

Except for the magic methods (those with underscores), there are 17 methods for sets.

- We have already covered intersection, union, difference, and symmetric difference above.
- You can execute `help(set)` for more details.

```
# methods of sets
# dir() returns a list

print(dir(set))
```

```
['__and__', '__class__', '__class_getitem__', '__contains__', '__delattr__', '__dir__',
```

```
print(dir(set)[-17:])
```

```
['add', 'clear', 'copy', 'difference', 'difference_update', 'discard', 'intersection',
```

### add()

It adds a new element to a set.

```
s = {1,2,3,4,5}
print(f'set s before using add(): {s}')

# add 99
s.add(99)

print(f'set s after using add() : {s}')
```

```
set s before using add(): {1, 2, 3, 4, 5}
set s after using add() : {1, 2, 3, 4, 5, 99}
```

## clear()

It removes all elements from the set, making it an empty set.

```
s = {1,2,3,4,5}
print(f'set s before using clear(): {s}')

# removes all elements
s.clear()

print(f'set s after using clear() : {s}')
```

```
set s before using clear(): {1, 2, 3, 4, 5}
set s after using clear() : set()
```

## copy()

It returns a new set with the same elements.

```
s = {1,2,3,4,5}
print(f'set s before using copy(): {s}')

# make a copy
s_copy = s1.copy()

print(f'set s after using copy() : {s}')
print(f'set s_copy : {s_copy}')
```

[Skip to main content](#)

```
set s before using copy(): {1, 2, 3, 4, 5}
set s after using copy() : {1, 2, 3, 4, 5}
set s_copy                 : {1, 2, 3, 4, 5}
```

## isdisjoint()

It checks if there are no common elements between the two given sets.

```
s1 = {1,2,3,4,5}
s2 = {3,4,5,6,7}

# False: 3,4,5 are common
print(f' s1 and s2 are disjoint: {s1.isdisjoint(s2)}' )
```

```
s1 and s2 are disjoint: False
```

```
s1 = {1,2,3,4,5}
s2 = {10,20,30}

# True: no common element
print(f' s1 and s2 are disjoint: {s1.isdisjoint(s2)}' )
```

```
s1 and s2 are disjoint: True
```

## issubset()

It checks if the first set is a subset of the second set.

```
s1 = {1,2,3,4,5}
s2 = {3,4,5,6,7}

# False: s1 is not subset of s2
print(f' s1 is subset of s2: {s1.issubset(s2)}' )
```

```
s1 is subset of s2: False
```

[Skip to main content](#)

```
s1 = {3,4,5}
s2 = {3,4,5,6,7}

# True: s1 is not subset of s2
print(f' s1 is subset of s2: {s1.issubset(s2)}' )
```

s1 is subset of s2: True

## isuperset()

It checks if the first set contains the second set.

```
s1 = {1,2,3,4,5}
s2 = {3,4,5,6,7}

# False: s1 does not contain s2
print(f' s1 is superset of s2: {s1.issuperset(s2)}' )
```

s1 is superset of s2: False

```
s1 = {1,2,3,4,5}
s2 = {3,4,5}

# True: s1 contains s2
print(f' s1 is superset of s2: {s1.issuperset(s2)}' )
```

s1 is superset of s2: True

## pop()

It removes a randomly selected element from the set.

- If the set is empty, an error message is raised.

```
s = {1,2,3,4,5}
print(f'set s before using pop(): {s}')

# removes a random element
removed_element = s.pop()

print(f'set s after using pop(): {s}')
print(f'removed element : {removed_element}'')
```

```
set s before using pop(): {1, 2, 3, 4, 5}
set s after using pop(): {2, 3, 4, 5}
removed element : 1
```

## remove()

It removes the specified element from the set.

- If the element does not exist in the set, an error message is raised.

```
s = {1,2,3,4,5}
print(f'set s before using remove(): {s}')

# removes 3
s.remove(3)

print(f'set s after using remove(): {s}'')
```

```
set s before using remove(): {1, 2, 3, 4, 5}
set s after using remove(): {1, 2, 4, 5}
```

## Iterations and Sets

We can use a *for* loop and iterate through values in the set operator to access each element of the set and perform operations on them.

- Indexes cannot be used as in *tuples* and *lists* since there is no ordering and indexing for sets.
- When you run a for loop through a set, the order of the values of state might be different.

[Skip to main content](#)

We can use a *for* loop to iterate through values in the set and access each element of the set to perform operations on them.

- Indexes cannot be used, as in tuples and lists, since there is no ordering and indexing for sets.
- When you run a *for* loop through a set, the order of the values of the set might be different.

```
# print state names in states set
states = {'Arizona', 'Oklahoma', 'Texas', 'Florida', 'California'}    # states is a set

for state in states:
    print(state)          # not in the same order that you see above
```

Texas  
Florida  
Oklahoma  
Arizona  
California

## Sets Debugging

- Each of the following short code contains one or more bugs.
- Please identify and correct these bugs.
- Provide an explanation for your answer.

## Question

```
letters = ['a', 'b', 'c']

print(letters[1])
```

### Solution

There is no indexing for sets.

## Question

[Skip to main content](#)

```
letters = {'a', 'b', 'c'}  
letters.append('d')  
print(letters)
```

### Solution ▼

Sets do not have an append() method; instead, they use the add() method.

## Question

```
letters = {'a', 'b', ['c', 'd']}  
print(letters)
```

### Solution ▼

A list cannot be an element of a set.

## Question

```
numbers = {1,2,3,4,'5'}  
print(sum(numbers))
```

### Solution ▼

'5' is a string and cannot be added to integers.

## Question

```
letters = {'a', 'b', 'c'}
```

[Skip to main content](#)

```
print(letters)
```

### Solution

The pop() method has no parameters.



## Strings Output

- Find the output of the following code.
- Please don't run the code before giving your answer.

## Question

```
letters = {'f'}
letters.add('k')
letters.clear()
letters.add('t')
letters.add('t')
letters.add('t')

print(letters)
```

► Show code cell output

## Question

```
numbers = [5, 8, 6, 1, 5, 5, 6, 7]
numbers = set(numbers)
print(sum(numbers))
```

► Show code cell output

## Question

[Skip to main content](#)

```
state = 'california'  
print(len(state))  
state_set = set(state)  
print(len(state_set))
```

► Show code cell output

## Question

```
letters = ['a', 'c', 'K', 'T', 't', 'A', 'C', 'a']  
letters = [char.lower() for char in letters]  
letters = set(letters)  
letters = list(letters)  
letters.sort()  
  
print(letters)
```

► Show code cell output

## Sets Code

- Please solve the following questions using Python code.

## Question

Identify the unique names in the list *names* below (not case-sensitive), and store them in a new list with dictionary order and capitalized.

```
names = ['Ben', 'Eli', 'Ben', 'Ian', 'Mia', 'Eva', 'Ana', 'Leo', 'Tim', 'liz', 'ben',
```

## Solution

► Show code cell content

[Skip to main content](#)

Remove names that start with a capital letter from the lists below.

- Store the unique names (not case-sensitive) in a new list, ensuring they are capitalized and ordered.

```
names1 = ['Ben', 'ELI', 'Ben', 'Ian', 'MIa', 'Eva', 'Ana', 'Leo', 'Tim', 'liz', 'ben',  
names2 = ['Max', 'Eli', 'Ben', 'Ian', 'Amy', 'mia', 'Eva', 'anA', 'Leo', 'Joe', 'Liz',
```

## Solution

▶ Show code cell content

# Chp-11 Dictionaries

- Learning Objectives
  - ..
  - ..

## Dictionaries

These are a more general form of lists. The indexes of lists are integers starting from 0, whereas the indexes, called keys in dictionaries, can be chosen from different types.

- Its elements are pairs of the form `key:value`.
- Dictionaries are created by using curly brackets, like sets, by using `key:value` pairs and `:` in between.
- Keys are like indexes, and values can be accessed by using square brackets in the form of `dict_name[key]`.
- Keys must be immutable, like strings, numbers, and tuples.
  - A list cannot be a key of a dictionary.
- Values can be any type, including strings, numbers, booleans, tuples, lists, and dictionaries.
- Dictionary pairs are ordered.
- If there are two pairs with the same key, then the later pair will overwrite the former one.

[Skip to main content](#)

- Since dictionaries can be modified, they have a large number of methods.
- The `len()` function returns the number of pairs.
- The built-in `dict()` function can be used to create dictionaries from structures that have pairs.
- `{}` is an empty dictionary.
  - An empty set is `set()`.

## Create Dictionaries

```
# empty dictionary
empty_dict = {}

print(f'Empty dictionary      : {empty_dict}')
print(f'Type of empty_dict    : {type(empty_dict)}')
```

```
Empty dictionary      : {}
Type of empty_dict   : <class 'dict'>
```

```
# empty dictionary with dict()
empty_dict = dict()

print(f'Empty dictionary      : {empty_dict}')
print(f'Type of empty_dict    : {type(empty_dict)}')
```

```
Empty dictionary      : {}
Type of empty_dict   : <class 'dict'>
```

```
# three pairs
# keys: 'Math', 'Chemistry', 'History'
# values: 80, 70, 65

grades = {'Math':80, 'Chemistry':70, 'History':65}

print(grades)
```

```
{'Math': 80, 'Chemistry': 70, 'History': 65}
```

```
# overwrite first mpair of math  
  
grades = {'Math':80, 'Chemistry':70, 'History':65, 'Math':100}  
  
print(grades)      # The value of 'Math' key is 100
```

```
{'Math': 100, 'Chemistry': 70, 'History': 65}
```

```
# values canbe lists, tuples  
  
grades = {'Math':[80, 90], 'Chemistry':(70, 100), 'History':65}  
  
print(grades)      # The value of 'Math' key is 100
```

```
{'Math': [80, 90], 'Chemistry': (70, 100), 'History': 65}
```

```
# lists can not be a key  
  
grades = {[ 'Math', 'Biology']:80, 'Chemistry':70, 'History':65}    # ERROR
```

## dict()

```
# assignments ----> dict  
grades = dict(Math=80, Chemistry=70, History=65)  
  
print(grades)
```

```
{'Math': 80, 'Chemistry': 70, 'History': 65}
```

```
# tuples of pairs ----> dict  
  
grades_tuple = ( ('Math', 80), ('Chemistry', 70), ('History', 65))  
grades = dict(grades_tuple)  
  
print(grades)
```

[Skip to main content](#)

```
{'Math': 80, 'Chemistry': 70, 'History': 6}
```

## len()

The `len()` function returns the number of pairs in a dictionary.

```
grades = {'Math':80, 'Chemistry':70, 'History':65}  
print(f'The number of pairs: {len(grades)}')
```

```
The number of pairs: 3
```

## in & not in

- `in` tests whether a given term is a key of a dictionary.
- `not in` tests whether a given term is not a key of a dictionary.
- Both of them return boolean values, True or False.

```
grades = {'Math':80, 'Chemistry':70, 'History':65}  
  
print(f' Math is      a key of grades: {"Math"    in grades}' )      # use " instead of '  
print(f' Math is not a key of grades: {"Math"   not in grades}' )
```

```
Math is      a key of grades: True  
Math is not a key of grades: False
```

```
grades = {'Math':80, 'Chemistry':70, 'History':65}  
  
print(f' Biology is      a key of grades: {"Biology"    in grades}' )      # use " instead of '  
print(f' Biology is not a key of grades: {"Biology"   not in grades}' )
```

```
Biology is      a key of grades: False  
Biology is not a key of grades: True
```

# Mutable

Unlike strings and tuples, and like lists, dictionaries are mutable, which means they can be modified.

- New pairs can be added, and existing pairs can be modified.

## Add a new pair

- Use square brackets in the form of `dict_name[new_key] = new_value`.

```
grades = {'Math':80, 'Chemistry':70, 'History':65}
print(f'grades dictionary before adding a new pair: {grades}')

# add the pair 'English':99
grades['English'] = 99

print(f'grades dictionary after adding a new pair: {grades}')
```

```
grades dictionary before adding a new pair: {'Math': 80, 'Chemistry': 70, 'History': 65}
grades dictionary after adding a new pair: {'Math': 80, 'Chemistry': 70, 'History': 65}
```

## Change a Value

- Use square brackets in the form of `dict_name[key] = new_value`.

```
grades = {'Math':80, 'Chemistry':70, 'History':65}
print(f'grades dictionary before changing a value: {grades}')

# change the value of 'Chemistry'
grades['Chemistry'] = 85

print(f'grades dictionary after adding a new pair: {grades}')
```

```
grades dictionary before changing a value: {'Math': 80, 'Chemistry': 70, 'History': 65}
grades dictionary after adding a new pair: {'Math': 80, 'Chemistry': 85, 'History': 65}
```

## Delete a pair

[Skip to main content](#)

```
grades = {'Math':80, 'Chemistry':70, 'History':65}
print(f'grades dictionary before deleting a value: {grades}')

# delete 'Chemistry':70 pair
del grades['Chemistry']

print(f'grades dictionary after deleting a pair: {grades}')
```

```
grades dictionary before deleting a value: {'Math': 80, 'Chemistry': 70, 'History': 65}
grades dictionary after deleting a pair: {'Math': 80, 'History': 65}
```

## Dictionary Methods

Except for the magic methods (the ones with underscores), there are 11 methods for dictionaries.

- You can run `help(dict)` for more details.

```
# methods of dictionaries
# dir() returns a list

print(dir(dict))
```

```
['__class__', '__class_getitem__', '__contains__', '__delattr__', '__delitem__', '__dir__',
 __eq__', '__format__', '__ge__', '__getattribute__', '__gt__', '__hash__', '__init__', '__le__',
 '__lt__', '__ne__', '__new__', '__reduce__', '__reduce_ex__', '__repr__', '__setattr__', '__sizeof__',
 '__str__', '__subclasshook__']
```

```
# non magic methods by using slicing
print(dir(dict)[-11:])
```

```
['clear', 'copy', 'fromkeys', 'get', 'items', 'keys', 'pop', 'popitem', 'setdefault', 'update']
```

### clear()

It removes all pairs from the dictionary, making it an empty dictionary.

```
grades = {'Math':80, 'Chemistry':70, 'History':65}
print(f'grades dictionary before using clear(): {grades}')

# remove all elements of grades dictionary
grades.clear()

print(f'grades dictionary after using clear() : {grades}')
```

```
grades dictionary before using clear(): {'Math': 80, 'Chemistry': 70, 'History': 65}
grades dictionary after using clear() : {}
```

## copy()

It returns a new dictionary with the same pairs.

```
grades = {'Math':80, 'Chemistry':70, 'History':65}
print(f'grades dictionary      : {grades}')

# remove all elements of grades dictionary
grades_copy = grades.copy()

print(f'grades_copy dictionary: {grades}')
```

```
grades dictionary      : {'Math': 80, 'Chemistry': 70, 'History': 65}
grades_copy dictionary: {'Math': 80, 'Chemistry': 70, 'History': 65}
```

## get()

It takes a key and a default value.

- If the given key exists in the dictionary, `get()` returns the corresponding value; otherwise, it returns the default value.
- Its purpose is to prevent errors for non-existing key-value pairs.

```
grades = {'Math':80, 'Chemistry':70, 'History':65}
print(f'grades dictionary      : {grades}')

# value corresponding to 'Math' key
val = grades.get('Math', 'Does Not Exist')      # Math key exist so get returns 80 not 'Does Not Exist'

print(f'The value of Math key: {val}')
```

```
grades dictionary      : {'Math': 80, 'Chemistry': 70, 'History': 65}
The value of Math key: 80
```

```
grades = {'Math':80, 'Chemistry':70, 'History':65}
print(f'grades dictionary      : {grades}')

# value corresponding to 'Biology' key
val = grades.get('Biology', 'Does Not Exist')      # Biology key does not exist so get returns 'Does Not Exist'

print(f'The value of Biology key: {val}')
```

```
grades dictionary      : {'Math': 80, 'Chemistry': 70, 'History': 65}
The value of Biology key: Does Not Exist
```

## items()

- It returns the pairs of a dictionary as a tuple in a data structure called dict\_items.

```
grades = {'Math':80, 'Chemistry':70, 'History':65}
print(f'Items: {grades.items()}')
```

```
Items: dict_items([('Math', 80), ('Chemistry', 70), ('History', 65)])
```

## keys()

- It returns the keys of a dictionary in a data structure called dict\_keys.

[Skip to main content](#)

```
grades = {'Math':80, 'Chemistry':70, 'History':65}  
print(f'Keys: {grades.keys()}')
```

Keys: dict\_keys(['Math', 'Chemistry', 'History'])

## pop()

- It removes a key-value pair for a given key and returns the value removed.

```
grades = {'Math':80, 'Chemistry':70, 'History':65}  
print(f'grades dictionary before using pop(): {grades}')  
  
# remove 'Chemistry':70 pair  
val_removed = grades.pop('Chemistry')  
  
print(f'grades dictionary after using pop(): {grades}')  
print(f'Removed value: {val_removed}')
```

grades dictionary before using pop(): {'Math': 80, 'Chemistry': 70, 'History': 65}  
grades dictionary after using pop(): {'Math': 80, 'History': 65}  
Removed value: 70

## popitem()

- It removes the last key-value pair and returns the key-value pair that is removed as a tuple.

```
grades = {'Math':80, 'Chemistry':70, 'History':65, 'English':100}  
print(f'grades dictionary before using pop(): {grades}')  
  
# remove  
tuple_removed = grades.popitem()  
  
print(f'grades dictionary after using pop(): {grades}')  
print(f'Removed pair: {tuple_removed}')
```

grades dictionary before using pop(): {'Math': 80, 'Chemistry': 70, 'History': 65, 'Eng  
grades dictionary after using pop(): {'Math': 80, 'Chemistry': 70, 'History': 65}

[Skip to main content](#)

## update()

- It adds pairs of the second dictionary to the first one.

```
dict1 = {'A':1, 'B':2, 'C':3}
dict2 = {'D':4, 'E':5, 'F':6}
print(f'dict1 before update: {dict1}')

dict1.update(dict2)

print(f'dict1 after update: {dict1}')
```

```
dict1 before update: {'A': 1, 'B': 2, 'C': 3}
dict1 after update: {'A': 1, 'B': 2, 'C': 3, 'D': 4, 'E': 5, 'F': 6}
```

## values()

- It returns the values of a dictionary in a data structure called dict\_values.

```
grades = {'Math':80, 'Chemistry':70, 'History':65}
print(f'Values: {grades.values()}')
```

```
Values: dict_values([80, 70, 65])
```

## Iterations and Dictionaries

You can use `keys`, `values`, or `(keys, values)` with *for* loops.

- `keys()`, `values()`, and `items()` methods return the keys, values, and (keys, values) respectively.
- Which one is better to use depends on what you need.

```
# use keys
grades = {'Math':80, 'Chemistry':70, 'History':65}

for key in grades.keys():
    print(key)
```

[Skip to main content](#)

Math  
Chemistry  
History

```
# use values
grades = {'Math':80, 'Chemistry':70, 'History':65}

for value in grades.values():
    print(value)
```

80  
70  
65

```
# use items for (key,value) pairs
grades = {'Math':80, 'Chemistry':70, 'History':65}

for key,value in grades.items():
    print(key,value)
```

Math 80  
Chemistry 70  
History 65

- You can also access the values by using the keys.

```
# use keys to access values
grades = {'Math':80, 'Chemistry':70, 'History':65}

for key in grades.keys():
    print(grades[key])
```

80  
70  
65

## Dictionary Comprehension

[Skip to main content](#)

- It is similar to list comprehensions.
- It is in the form of: `{expression for item in list}`
  - Here, the `expression` represents the `key:value` pairs of the dictionary being constructed.
- An `if` statement can also be included in a dictionary comprehension in the form of: `[expression for item in list if condition]`

## Example

- The following code constructs a dictionary with `key:value` pairs where keys are the numbers between 1 and 5, and values are the cubes of the keys.

```
cube_dict = {i:i**3 for i in range(1,6)} # i is a key, i**3 is a value in the expression  
print(cube_dict)
```

```
{1: 1, 2: 8, 3: 27, 4: 64, 5: 125}
```

## Examples

### Word Lengths

Write a program that constructs a dictionary such that:

- keys are words in the text below.
- values are the lengths of the words.
- Example: ('Python', 6) is a pair

```
text = """" Imyep jgsqewt okbxsq seunh many rkx vmysz ndpoz may vxabckewro topfd tqkj ue  
....
```

### Solution:

[Skip to main content](#)

```

words = text.split()          # words in text
word_dict = {}                # empty dictionary
for i in words:               # i is a word
    word_dict[i] = len(i)      # add a pair: key is i, value is len(i)
print(word_dict)

```

```
{'Imyep': 5, 'jgsqewt': 7, 'okbxsq': 6, 'seunh': 5, 'many': 4, 'rkx': 3, 'vmysz': 5, 'n
```

## Count Characters

Write a program which constructs a dictionary such that:

- keys are the characters in the *text* below.
- values are the number of the occurrences of each character in the *text*.

```
text = """" Imyep jgsqewt okbxsq seunh many rkx vmysz ndpoz may vxabckewro topfd tqkj u
```

### Solution-1:

- By using *count()* method of strings.

```

count_dict = {}                      # empty dictionary
for char in text:                   # char is a character
    count_dict[char] = text.count(char) # add a pair: key is i, value is the occurrence
print(count_dict)

```

```
{' ': 1301, 'I': 2, 'm': 234, 'y': 220, 'e': 355, 'p': 185, 'j': 193, 'g': 201, 's': 23
```

### Solution-2:

- Without using *count()* method of strings.
- char will represent each character in *text*.

[Skip to main content](#)

- 'a' is not a key yet, so the pair ('a', 1) will be created.
- When char is the second 'a',
  - Since 'a' is already a key, only its value will be increased by one.
  - The pair ('a', 1) becomes ('a', 2).

```
count_dict = {}                                # empty dictionary

for char in text:
    if char not in count_dict.keys():
        count_dict[char] = 1                  # for the first occurrence of char the value is
  # pair is created: (char,1)
    else:
        count_dict[char] +=1                # char is already a key means it is repeated,
  # so we increment the value by 1

print(count_dict)
```

```
{' ': 1301, 'I': 2, 'm': 234, 'y': 220, 'e': 355, 'p': 185, 'j': 193, 'g': 201, 's': 23}
```

## Dictionaries Debugging

- Each of the following short code contains one or more bugs.
- Please identify and correct these bugs.
- Provide an explanation for your answer.

## Question

```
{'A'=1, 'B'=2, 'C'=3}
```

### Solution

For dictionaries, `=` should be replaced with `:` to associate keys with values.

## Question

[Skip to main content](#)

### Solution

my\_dict('C') must be my\_dict['C'].

## Question

```
my_dict = ['A':70, 'B':90, 'C':75, 'D':80]
my_dict['C']
```

### Solution

To construct a dictionary in Python, curly braces `{}` must be used instead of square brackets `[]`.

## Question

```
stock_dict = {'day-1':{'High': 70, 'Low':62, 'Close':65},
              'day-2':{'High': 68, 'Low':60, 'Close':63},
              'day-3':{'High': 71, 'Low':65, 'Close':67},
              'day-4':{'High': 70, 'Low':62, 'Close':65},
              'day-5':{'High': 73, 'Low':65, 'Close':70},
              'day-6':{'High': 75, 'Low':69, 'Close':73}
            }

for i in [1,3]:
    print(stock_dict['day-'+i]['Low'])
```

### Solution

The counter `i` takes integer values, so it cannot be directly concatenated with `'day-'`. To resolve this issue, use `'day-' + str(i)` instead.

## Dictionaries Output

[Skip to main content](#)

- Please don't run the code before giving your answer.

## Question

```
dict_exp = {'table':1, 'chair':2, 'washer':3}  
  
for i in dict_exp.keys():  
    print(i[-2])
```

► Show code cell output

## Question

```
dict_exp = {'table':[1,5,9], 'chair':[10,20,30], 'washer':[2,7,4]}  
total = 0  
  
for i in dict_exp.values():  
    total += i[1]  
  
print(total)
```

► Show code cell output

## Question

```
dict_exp = {'table':[1,5,9], 'chair':[4,6,8], 'washer':[2,7,4]}  
total = 0  
  
for i in dict_exp.values():  
    for j in i:  
        total += i[1]  
  
print(total)
```

► Show code cell output

## Question

[Skip to main content](#)

```
dict_exp = {'table':[1,5,9], 'chair':[4,6,8], 'washer':[2,7,4]}\ntotal = 0\n\nfor i in dict_exp.values():\n    for j in i:\n        total += j\n\nprint(total)
```

► Show code cell output

## Question

```
stock_dict = {'day-1':{'High': 70, 'Low':62, 'Close':65},\n             'day-2':{'High': 68, 'Low':60, 'Close':63},\n             'day-3':{'High': 71, 'Low':65, 'Close':67},\n             'day-4':{'High': 70, 'Low':62, 'Close':65},\n             'day-5':{'High': 73, 'Low':65, 'Close':70},\n             'day-6':{'High': 75, 'Low':69, 'Close':73}\n            }\n\nfor i in [1,3]:\n    print(stock_dict['day-'+str(i)]['Low'])
```

► Show code cell output

## Question

```
stock_dict = {'day-1':{'High': 70, 'Low':62, 'Close':65},\n             'day-2':{'High': 68, 'Low':60, 'Close':63},\n             'day-3':{'High': 71, 'Low':65, 'Close':67},\n             'day-4':{'High': 70, 'Low':62, 'Close':65},\n             'day-5':{'High': 73, 'Low':65, 'Close':70},\n             'day-6':{'High': 75, 'Low':69, 'Close':73}\n            }\ntotal = 0\n\nfor i,j in stock_dict.items():\n    total += int(i[-1])\n\nprint(total)
```

► Show code cell output

[Skip to main content](#)

## Question

```
stock_dict = {'day-1':{'High': 70, 'Low':62, 'Close':65},  
             'day-2':{'High': 68, 'Low':60, 'Close':63},  
             'day-3':{'High': 71, 'Low':65, 'Close':67},  
             'day-4':{'High': 70, 'Low':62, 'Close':65},  
             'day-5':{'High': 73, 'Low':65, 'Close':70},  
             'day-6':{'High': 75, 'Low':69, 'Close':73}  
         }  
total = 0  
  
for i,j in stock_dict.items():  
    total += j['High']//10  
  
print(total)
```

► Show code cell output

## Question

```
my_dict = {200:'NY', 100:'NJ', 400:'TX', 300:'CA'}  
  
for i in my_dict.keys():  
    if i>188:  
        print(my_dict[i][1])
```

► Show code cell output

## Question

```
mydict = {'A':(10,100), 'B':(20, 200), 'C':(30,300)}  
  
for i,j in mydict.values():  
    print(i,j)
```

► Show code cell output

## Dictionaries Code

- Please solve the following questions using Python code.

[Skip to main content](#)

## Question

Write a program that constructs a dictionary using two lists provided below, where the keys are taken from loc\_list and the values are taken from player\_list.

- Each pair in the dictionary corresponds to a key-value pair with elements from the same index in the lists.
- Use a for loop for this construction.

```
loc_list = ['Boston', 'Portland', 'Brooklyn', 'Dallas']
player_list = ['Tatum', 'Lillard', 'Durant', 'Doncic']
```

## Solution

▶ Show code cell content

## Question

Write a program that constructs a dictionary using the given list below, where the keys are taken from state\_list and the values represent the number of characters in each state name.

- Each pair in the dictionary corresponds to a state name and the number of characters in its name.
- Example: {'California': 10}

```
state_list = ['Utah', 'Nevada', 'Florida', 'Texas', 'Oklahoma', 'Washington', 'Colorado']
```

## Solution-1

▶ Show code cell content

```
print(state_dict)
```

```
{'Utah': 4, 'Nevada': 6, 'Florida': 7, 'Texas': 5, 'Oklahoma': 8, 'Washington': 10, 'Colorado': 11}
```

## Solution-2

[Skip to main content](#)

```
print(state_dict)
```

```
{'Utah': 4, 'Nevada': 6, 'Florida': 7, 'Texas': 5, 'Oklahoma': 8, 'Washington': 10, 'Co
```

## Question

For the given dictionary below, display 'Close' values for each day.

```
stock_dict = {'day-1':{'High': 70, 'Low':62, 'Close':65},  
             'day-2':{'High': 68, 'Low':60, 'Close':63},  
             'day-3':{'High': 71, 'Low':65, 'Close':67},  
             'day-4':{'High': 70, 'Low':62, 'Close':65},  
             'day-5':{'High': 73, 'Low':65, 'Close':70},  
             'day-6':{'High': 75, 'Low':69, 'Close':73}  
         }
```

## Solution

► Show code cell content

## Question

Use the *stock\_dict* to create a list consisting of the 'Close' values."

## Solution

► Show code cell content

## Question

Use *stock\_dict* to construct a dictionary where the keys are days (Day-1, Day-2, ...) and the values represent the difference between 'High' and 'Low' values.

## Solution

[Skip to main content](#)

## Question

Use the `stock_dict` to construct a list consisting of values 'Increasing' or 'Decreasing' based on the difference between two consecutive day's 'Close' values.

- Note that there is no 'Increasing' or 'Decreasing' value for Day-1, as the value before Day-1 is not provided.

## Solution

▶ Show code cell content

## Question

Construct a dictionary using a for loop, where the keys are the names in `names_list` and the values are the last characters of the corresponding names.

- Example: key = `ashley`, value = `y`

```
names_list = ['ashley', 'michael', 'jack', 'taylor', 'tim', 'robert', 'joseph']
```

## Solution

▶ Show code cell content

## Question

Construct a dictionary using a for loop, where the keys are the first names in `names_list` and the values are the corresponding last names.

- Example: key='Michael', value='Jordan' for 'Michael Jordan' in `fullnames_list`.

```
fullnames_list = ['Michael Jordan', 'Larry Bird', 'Jason Tatum', 'Lebron James', 'Jimmy
```

## Solution

[Skip to main content](#)

## Question

Use a list comprehension to store the last two characters of each name in a list

```
grades_dict = { 'mike' : {'Math':90, 'History':88, 'Science':73},  
                'jack' : {'Math':77, 'History':74, 'Science':52},  
                'tim' : {'Math':98, 'History':35, 'Science':46},  
                'liz' : {'Math':65, 'History':55, 'Science':81},  
                'aria' : {'Math':76, 'History':99, 'Science':69}}
```

## Solution

► Show code cell content

## Question

Use a for loop to store all Math grades in a list.

## Solution

► Show code cell content

## Question

Use a for loop to find the name of the student with the largest Science grade and print both the name and the corresponding grade.

## Solution

► Show code cell content

## Question

Write a function whose parameters are two numbers.

- It returns a dictionary with
  - keys : sum, difference, product, quotient

[Skip to main content](#)

- If the second number is zero, then the quotient is labeled as 'DNE' (Does Not Exist)
- Example: inputs `6, 2`
  - Output: `{'sum': 8, 'difference':4, 'product':12, 'quotient':3.0}`

## Solution

▶ Show code cell content

```
print(operation_func(6, 2))
print(operation_func(6, 0))
```

```
{'sum': 8, 'difference': 4, 'product': 12, 'quotient': 3.0}
{'sum': 6, 'difference': 6, 'product': 0, 'quotient': 'DNE'}
```

