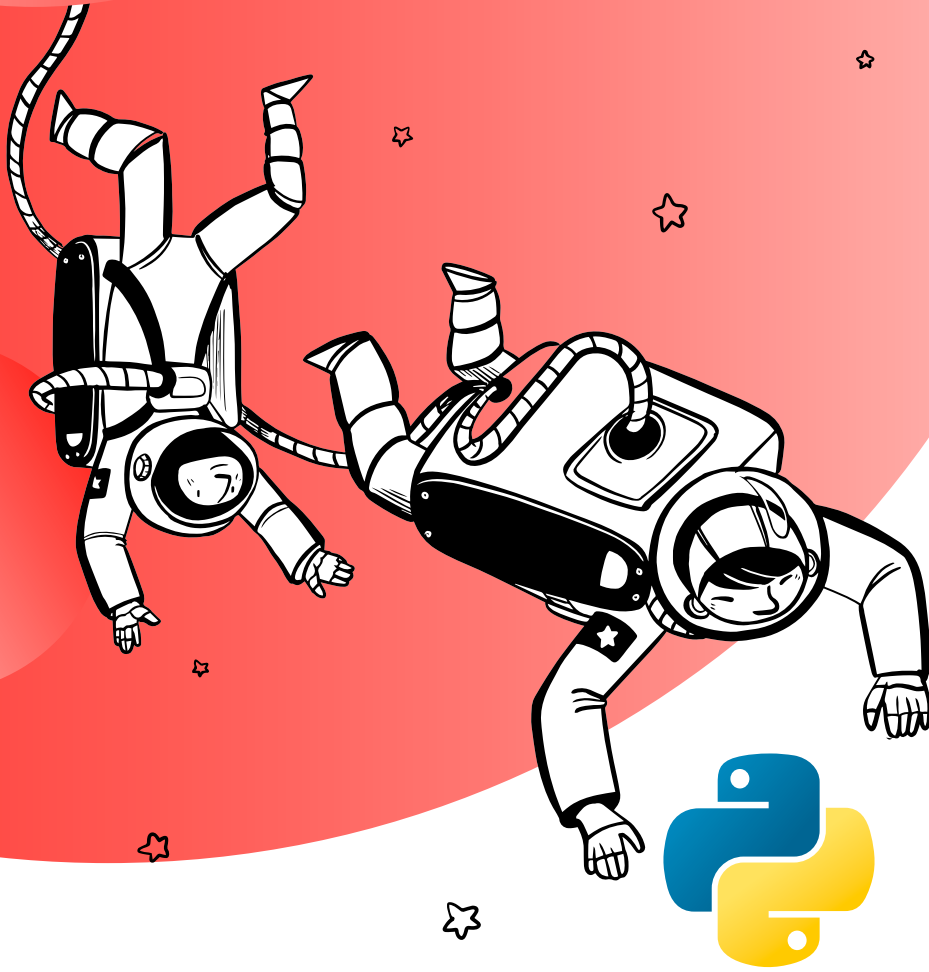


atoti.

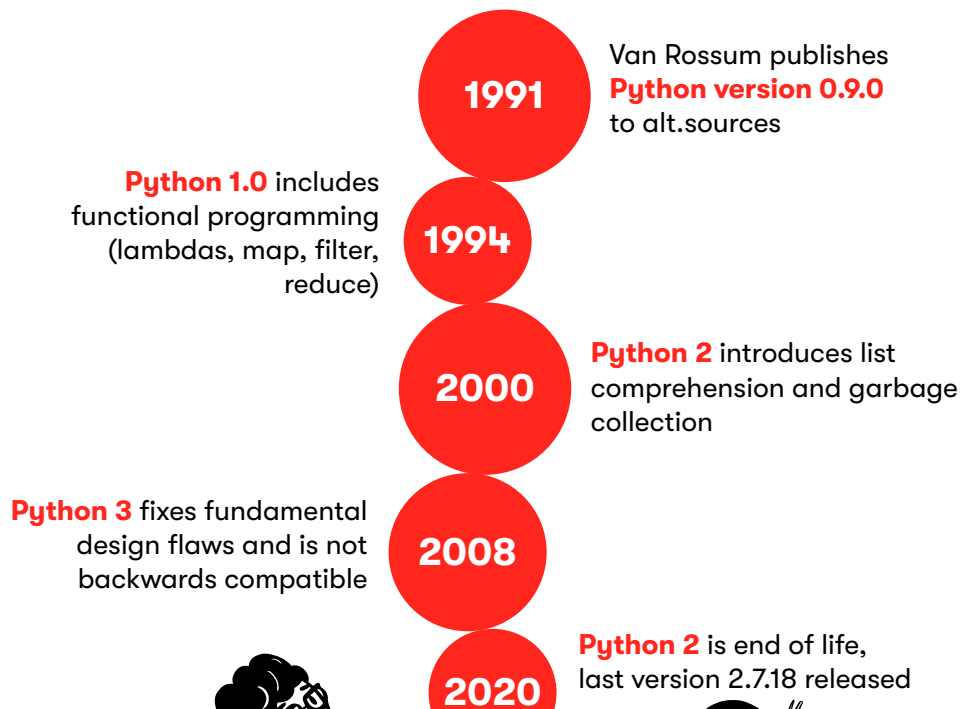


What is Python?

In data science

Python is an easy-to-learn programming language with a long history.

Python History



Timeline from <https://python.land/python-tutorial/python-history>.
Go to their webpage for full history of Python

How easy is it?

Let's compare with other languages using "Hello World".



```
using System;

namespace HelloWorld
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Hello World!");
        }
    }
}
```

```
#include <iostream>
using namespace std;

int main() {
    cout << "Hello World!";
    return 0;
}
```



```
public class Main {
    public static void main(String[] args) {
        System.out.println("Hello World");
    }
}
```

```
fun main() {
    println("Hello World")
}
```



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

Intuitive syntax

```
"Hello World!"
```



Try them out @ [w3schools.com](https://www.w3schools.com)

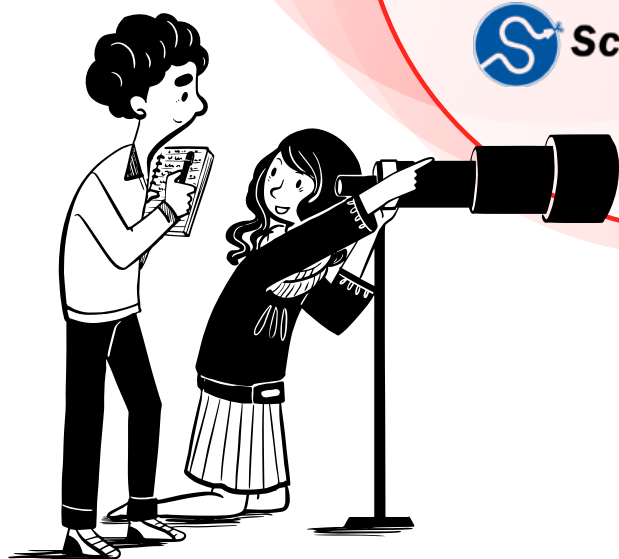
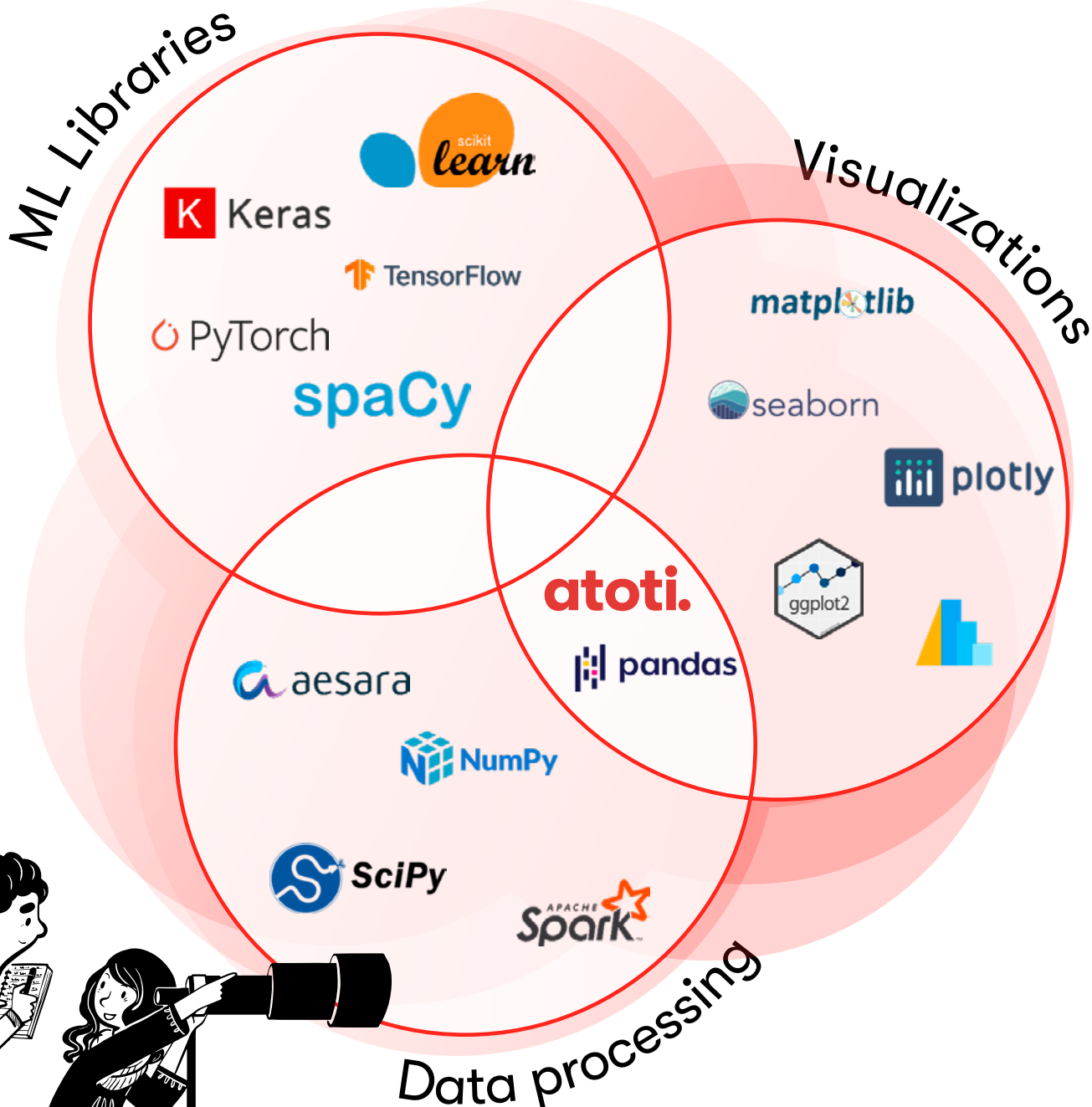


- General-purpose language for development and deployment
- Used by developers, data engineers and data scientists
- Less constraints on memory usage, therefore it can easily handle large data
- Highly interactive visualizations and dashboarding tools such as atoti, Plotly and Seaborn.

- Open-source statistical programming language
- Used by statisticians, analysts and data scientists for statistical analysis and graphical techniques
- Computes everything in memory, therefore it's limited by RAM size.
- Highly customizable static graphing tools such as ggplot2

Why is Python popular in Data Science?

Because it has amazing libraries!

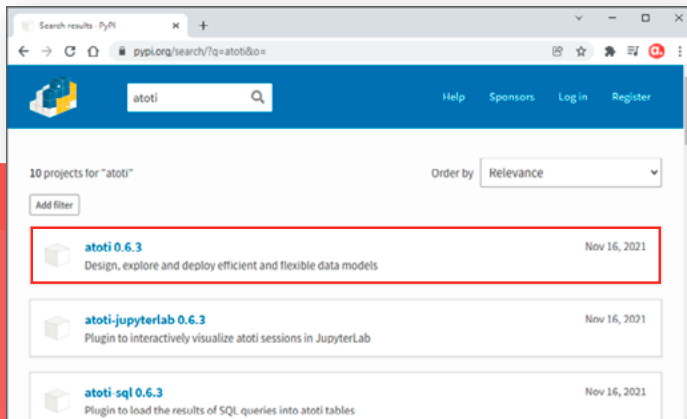


* Above are examples of Python libraries available for use but not limited to these.

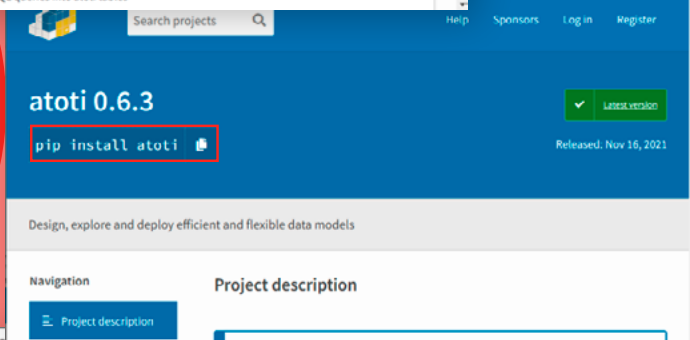
How do we get libraries?

1. Using pip

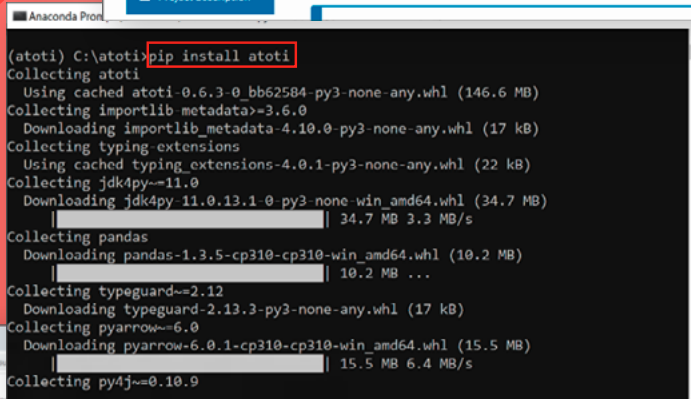
A repository of software for the Python programming language. Install package from <https://pypi.org/>.

**1**

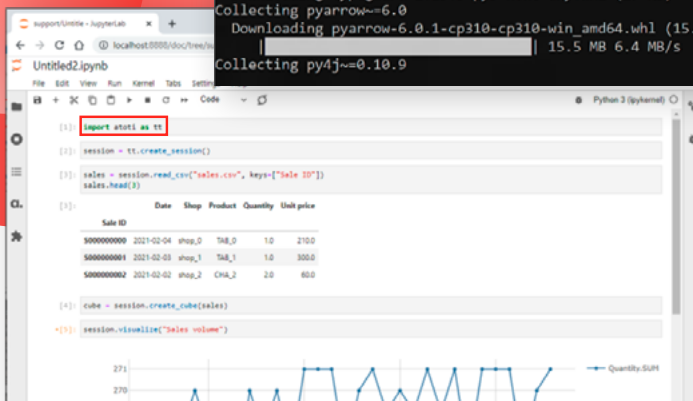
Search and select Python library

**2**

Copy pip install command

**3**

Run command in console

**4**

Import library and start using

2. Using conda

An open source package management system and environment management system.
Install package from <https://repo.anaconda.com/>.

The screenshot shows the 'Conda package' section of the atoti documentation. It includes a sidebar with navigation links and a main content area with installation instructions. Three numbered callouts are overlaid on the page:

- 1** Pre-requisite: Anaconda or Miniconda installed
- 2** Ensure channel for library is added**
- 3** Run "conda install" for package in command prompt

The documentation content includes the following sections and code snippets:

- Conda package**
Install [Miniconda 64-bit](#) or [Anaconda 64-bit](#).
- Note**
Conda 64-bit is required since recent versions of some packages are not available with Conda 32-bit.
- Add the conda-forge channel:**

```
conda config --add channels conda-forge
```
- Add the atoti channel:**

```
conda config --add channels https://conda.atoti.io
```
- Create a new Conda environment:**

```
conda create --name atoti
```
- Activate it:**

```
conda activate atoti
```
- Install atoti and its JupyterLab extension:**

```
conda install atoti atoti-jupyterlab python
```

* Conda installs packages which may contain software written in any language.

** "conda-forge" channel is free for all to use.

Download conda cheatsheet from <https://docs.conda.io/projects/conda/en/latest/user-guide/cheatsheet.html>.

Virtual environment

Create an isolated environment for Python projects with its own version of Python interpreter, libraries and scripts installed into it.



Create environment

```
python3 -m venv myenv
```

Ensure using Python 3 is used instead of Python 2

Activate environment

- Unix / MacOS

```
source myenv/bin/activate*
```

*Use activate.csh/activate.fish for csh/fish shell

- Windows

```
myenv\Scripts\activate.bat
```

Deactivate environment

```
deactivate
```

Remove environment

Close environment and delete directory



Create environment

```
conda create --name myenv python=3.8 atoti scipy
```

Create env
with specific
Python version

Install multiple
packages

Activate environment

```
conda activate myenv
```

Deactivate environment

```
conda deactivate
```

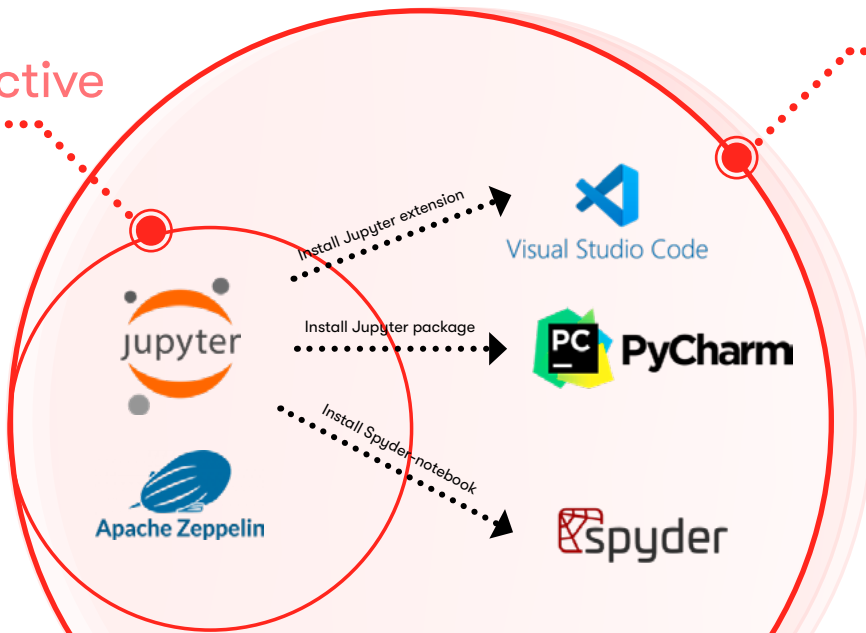
Remove environment

```
conda remove --name myenv --all
```



Integrated development environment (IDE) vs Code editor

Interactive



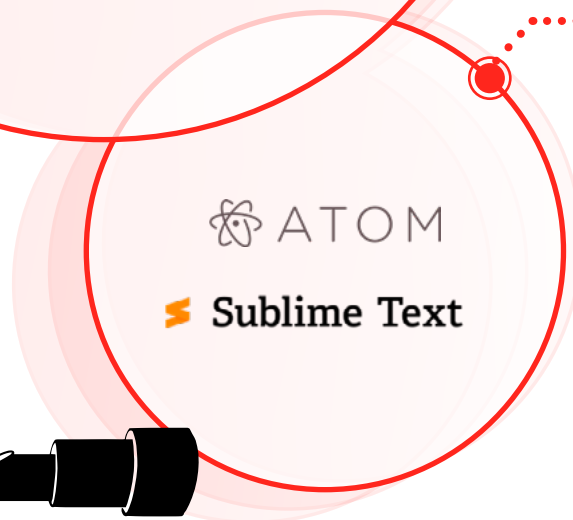
IDE

A set of tools that makes coding easier: text editor, compiler, build, debugging, auto linting function, auto-completion, etc

Extract codes from Jupyter notebook to .py files for production:
`jupyter nbconvert --to script my_notebook.ipynb`

Code editor

Text editors with powerful built-in features and specialized functionalities to simplify and accelerate code editing process



Dependency management

Dependencies are all the software components required by the project to work as intended.



1. Set up virtual environment
2. Install required modules
3. `pip freeze > requirements.txt`

To install dependencies in new env: `pip install -r requirements.txt`

<https://docs.python.org/3/tutorial/venv.html>



1. Set up virtual environment
2. Install required modules
3. `conda list -e > requirements.txt`

To install dependencies in new env:

`conda install -n <env_name> requirements.txt`

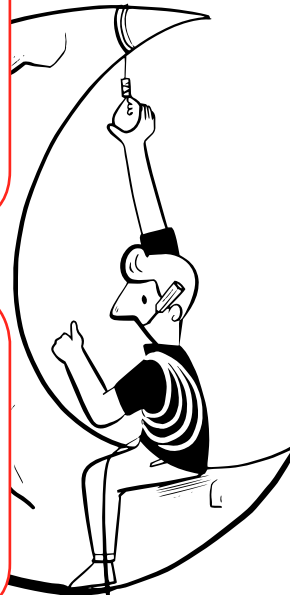


1. `poetry init *`
2. `poetry add <package>`
3. `pyproject.toml` file updated with dependency
4. sub-dependency version updated in `poetry.lock`

To install dependencies in new env:

1. `pyproject.toml` present
2. `poetry install`

* To use poetry to control an existing project.
Use "`poetry new <package name>`" to start new project.



Base Type

Category	Data types	Description	Examples
Numeric Types	int	whole number, positive or negative, without decimals, of unlimited length	0, 1, -327
	float	number, positive or negative, containing one or more decimals	0.0, 2.8, -312.67, 1.102e+12 x10 ¹²
Boolean Type	str	<p>immutable sequences of Unicode characters</p> <p>String surrounded by single or double quotes.</p> <p>Triple quotes for multiline string.</p> <p>Backslash to escape characters.</p> <p>\' ~ single quote \\ ~ backslash \b ~ backspace \n ~ new line \r ~ carriage return \t ~ tab</p>	<p>"Hello world", 'Hello world',</p> <p>""" Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua."""</p> <p>'It isn\'t where you came from. \nIt's where you're going that counts. '</p>
Boolean Type	bool		True, False
Binary Types	bytes	immutable sequences of single bytes	<pre>[1]: b"""Line 1. Line 2. """ [1]: b'Line 1.\nLine 2. \n'</pre>
	bytearray	mutable counterpart to bytes objects	<pre>[1]: byte_arr=bytearray("""Line 1. Line 2.""", "utf8") print(byte_arr) bytearray(b'Line 1.\nLine 2.')</pre> <pre>[2]: del byte_arr[0:8] print(byte_arr) bytearray(b'Line 2.')</pre>
	memoryview	Access the internal data of an object without copying, unlike bytes/str	<pre>[1]: mv = memoryview(byte_arr) print(bytes(mv[0:7])) b'Line 2.'</pre> <pre>[2]: mv[6] = 33 print(byte_arr) bytearray(b'Line 2!')</pre>

Base Type

Category	Data types	Description	Examples
Sequence Types	list	Container holding comma-separated values between square brackets. Mutable.	<code>["apple", "banana", "cherry"],</code> <code>["Friday", datetime.date(2022, 1, 14), 2022, 'Jan', 14]</code>
	tuple	Container holding comma-separated values (parentheses are optional). Immutable.	<code>("apple", "banana", "cherry"),</code> <code>date_tuple = "Friday", datetime.date(2022, 1, 14), 2022, "Jan", 14</code>
	range	Function returning sequence of numbers, starting from 0 and increments by 1 by default. <code>range(start, stop, step)</code>	<code>range(6), range(1,6)</code>
Mapping Types	dict	Ordered collection of data stored in key-value pairs. Key is unique.	<code>date_dict = {</code> <code> "day_of_week": "Friday",</code> <code> "date": datetime.date(2022, 1, 14),</code> <code> "day": 14,</code> <code> "month": "Jan",</code> <code> "year": 2022</code> <code>}</code>
Set Types	set	Unordered collection of data that is mutable, iterable and has unique elements.	<code>{"apple", "banana", "cherry"},</code> <code>{"Friday", datetime.date(2022, 1, 14), 2022, "Jan", 14}</code>
	frozen-set	Freeze iterable objects, making them unchangeable.	<code>frozenset({"apple", "banana", "cherry"})</code>

Type conversions

From type → To type	Examples
str → int	<code>int("10") → 10</code>
float → int	<code>int(10.23) → 10</code>
anything → str	<code>str(10.10) → '10.1'</code> , <code>str(1==1) → 'True'</code>
ASCII code → char	<code>chr(36) → '\$'</code>
char → ASCII code	<code>ord('\$') → 36</code>
str → list	<code>list("atoti") → ['a', 't', 'o', 't', 'i']</code> <code>"The quick brown fox jumped over the lazy dog".split()</code> <code>→ ['The', 'quick', 'brown', 'fox', 'jumped', 'over', 'the', 'lazy', 'dog']</code>
dict → list	<code>d = {"day": 14, "month": "Jan", "year": 2022}</code> <code>list(d.items()) → [('day', 14), ('month', 'Jan'), ('year', 2022)]</code> <code>list(d.keys()) → ['day', 'month', 'year']</code> <code>list(d.values()) → [14, 'Jan', 2022]</code> <code>[(k, v) for k, v in d.items()]</code> <code>→ [('day', 14), ('month', 'Jan'), ('year', 2022)]</code> <code>list(zip(d.keys(), d.values()))</code> <code>→ [('day', 14), ('month', 'Jan'), ('year', 2022)]</code> <code>list(map(list, d.items()))</code> <code>→ [('day', 14), ('month', 'Jan'), ('year', 2022)]</code>
list → dict	<code>dict([('day', 14), ('month', 'Jan'), ('year', 2022)])</code> <code>→ {"day": 14, "month": "Jan", "year": 2022}</code>
list → set	<code>set(['a', 't', 'o', 't', 'i']) → {'a', 'i', 'o', 't'}</code> Only unique items in set, only 1 't' in result.
list → string	<code>" ".join(['The', 'quick', 'brown', 'fox', 'jumped', 'over', 'the', 'lazy', 'dog'])</code> <code>→ 'The quick brown fox jumped over the lazy dog'</code>

Strings

Print

Code	Results
<code>print("Hello")</code>	Hello
<code>print("Hello", "world")</code>	Hello world
<code>print("Hello", "world", sep="...")</code>	Hello...world
<code>print("Hello", "world", sep="...", end="!")</code>	Hello...world!
<code>print("Hello")</code> # each print uses a newline <code>print("world")</code>	Hello world
<code>print("Hello", end=" ")</code> # without newline <code>print("world")</code>	Hello world

Concatenation

Code	Results
<code>a = "Hello"</code> <code>b = "world"</code> <code>year = 2022</code> <code>print(a + " " + b + " " + str(year))</code> # cast numeric value to str before concatenation	Hello world 2022

Formatting

Code	Results
<code>"Price: \${price:,.2f}".format(price = 1000)</code>	'Price: \$1,000.00'
<code>"{0}, {1}, {2}".format("apple", "banana", "cherry")</code>	'apple, banana, cherry'
<code>"{2}, {1}, {0}".format(*"abc")</code> # unpacking argument sequence	'c, b, a'
<code>"{:<30}".format("left aligned")</code> # with * as a fill char or <code>"left aligned".ljust(30, "*")</code>	'left aligned*****'
<code>"{:>30}".format("right aligned")</code> or <code>"right aligned".rjust(30)</code>	' right aligned'

Strings

Formatting

Code	Results
<pre> "{}".format("centered") or "centered".center(30) </pre>	' centered '
<pre> name = "Jane" f"Hello, {name}!" </pre>	'Hello, Jane!'
<pre> "198".zfill(5) # zero pad string </pre>	'00198'
<pre> import datetime d = datetime.datetime(2022, 1, 14, 10, 15, 58) "{:%Y-%m-%d %H:%M:%S}".format(d) </pre>	'2022-01-14 10:15:58'
<p>Change case:</p> <pre> "apple".capitalize() # Converts the first character to upper case "APPLE".casefold() # Converts string into lower case "APPLE".lower() # Converts a string into lower case "apple".upper() # Converts a string into upper case "Apple".swapcase() # Swaps cases, lower case becomes upper case and vice versa "apple".title() # Converts the first character of each word to upper case </pre>	'Apple' 'apple' 'apple' 'APPLE' 'aPPLE' 'Apple'

String functions

Code	Results
<pre> "apple".startswith("a") "apple".endswith("e") " apple ".strip() "apple".count("p") "apple".index("l") "apple".find("l") "apple".find("x") </pre>	True → string started with "a" True → string ended with "e" 'apple' → remove whitespace before & after string 2 → 2 "p"s are found in string 3 → index of "l" is 3 3 → "l" found at index 3 (index starts with 0) -1 → "x" not found in string

Functional modules

Functions

Code

Function name argument default value Optional

```
def greet(name="everybody", myname=None):
    print(f"Hello, {name}!")
    if myname:
        print(f"I am {myname}.")
```

Indentation

```
greet()
greet("Jane")

greet("Jane", "Thomas")
```

Results

```
Hello, everybody!
Hello, Jane!

Hello, Jane!
I am Thomas.
```

Exception handling

Code

```
try:
    print("Hello " + 123)
except Exception as e:
    print(f"Error occurred: {str(e)}")
finally:
    print("Gets printed no matter what")
```

```
num1 = 1
num2 = "1"
if num1 != num2:
    raise Exception(f"num1 ({num1}) is not equal to num2 ({num2})")
```

Results

Error occurred: can only concatenate str (not "int") to str
Gets printed no matter what

Exception: num1 (1) is not equal to num2 (1)

Functional modules

Modules/Names Import

Code	Results
import datetime → access datetime module datetime.datetime.now() datetime.date.today() datetime.date.today() + datetime.timedelta(days=10)	datetime.datetime(2022, 1, 17, 10, 52, 43, 795534) datetime.date(2022, 1, 17) datetime.date(2022, 1, 27)
from datetime import date, datetime, timedelta → direct function access from datetime import date as dt → give function an alias datetime.now() date.today() date.today() + timedelta(days=10) dt.today()	datetime.datetime(2022, 1, 17, 10, 52, 43, 795534) datetime.date(2022, 1, 17) datetime.date(2022, 1, 27) datetime.date(2022, 1, 17)
def greet(name): print(f"Hello, {name}!") → define function and saved in hello.py in same directory as main prog from hello import greet greet("Thomas")	Hello, Thomas!

Iterative looping

Code	Results
<div style="display: flex; align-items: center;"> <div style="writing-mode: vertical-rl; transform: rotate(180deg); margin-right: 5px;">Indentation</div> <div style="border: 1px solid black; padding: 5px; margin-left: 10px;"> <pre>for var in sequence: Do something</pre> </div> </div> for num in range(1,6): print(num, end=",")	 1,2,3,4,5,
fruits = ["apple", "banana", "cherry"] for f in fruits: if f == "banana": break → exit from the loop print(f)	 apple
i = 1 → initialize variable with value while i <= 10: i += 1 → conditional iteration; stops when i becomes greater than 10 print(i)	 16

Strings

Conditional

Code

```

basket1 = ["apple", "banana", "cherry"]
basket2 = ["kiwi", "pineapple", "orange"]

if "kiwi" in basket1:
    print("Found kiwi in basket1!")
elif "kiwi" in basket2: → equivalent of "else if" condition
    if all(x in basket2 for x in ["kiwi", "orange"]): If true
        print("Kiwi and orange found in basket2!")
    else: → default action when conditions on the same level are not met
        print("Kiwi not found in basket1 nor basket2.")

```

Indentation

If true

If false

Results

Kiwi and orange found in basket2!

Read/Write files

Code

Python's built-in func

```

f = open("file.txt", "w") ← w: create a file if it does not exist
                             x: create a file, returns error if file already exists
f.write("Existing content will be cleared!")
f.close()

f = open("file.txt", "a") ← a: append mode
f.write("\nThis will be appended to existing content! ")
f.close() To append as newline

f = open("file.txt", "r") ← r: read-only
print(f.read())

```

Results

Existing content will be cleared! This will be appended to existing content!

Refer to the following sites for more Python examples:

<https://www.w3schools.com/python/>

<https://www.geeksforgeeks.org/>

Check out <https://www.atoti.io/> to find out how to create a BI analytics platform in Python.

Stay tuned to <https://www.youtube.com/atoti> for useful tutorials.