Natural Language Processing
NLP_CLT_1st_APR_27st_2025

Eng. Maytham Ghanoum

Artificial Intelligence & Deep Learning Specialist
MTN Syria – SCS – SVU CLT
+963947222064 - +963982018359
https://www.linkedin.com/in/maytham-ghanoum-697aa5207/
https://www.facebook.com/maytham.ghanoum

Python Programming Languages

Python Programming Languages:

Python is a high-level, general-purpose programming language that has become one of the most popular and versatile languages in the world. Known for its simplicity, readability, and vast ecosystem of libraries, Python is widely used across various domains, including web development, data science, artificial intelligence, machine learning, and, most notably, bioinformatics.

Python Programming Languages:

Features of Python:

Readability and Simplicity:

Python’s clean and straightforward syntax is designed to be easy to read and write, making it an ideal language for beginners and experienced developers alike.

Interpreted Language:

Python is an interpreted language, meaning that code is executed line-by-line, which facilitates rapid prototyping, testing, and debugging.

Cross-Platform Compatibility:

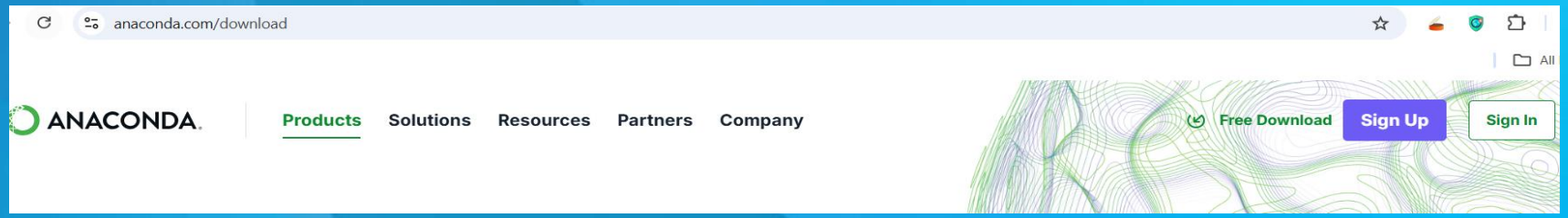
Python is compatible with various operating systems, including Windows, macOS, and Linux, allowing developers to write code that runs seamlessly across different platforms.

Extensive Library Support:

One of Python's greatest strengths is its extensive collection of libraries and frameworks. These libraries simplify complex tasks such as data analysis (e.g., NumPy, pandas), scientific computing (e.g., SciPy), and Deep learning (e.g., TensorFlow, PyTorch).

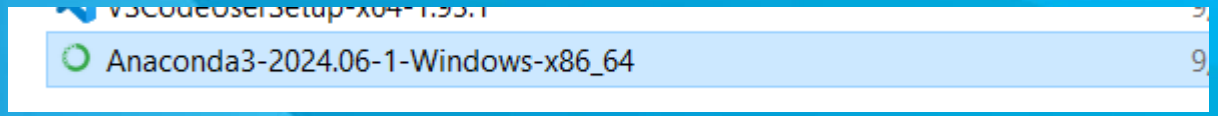
Python Programming Languages:
Preparing of Python Environment:

The Best Environment for Python called Anaconda
1- Go to https://www.anaconda.com/download

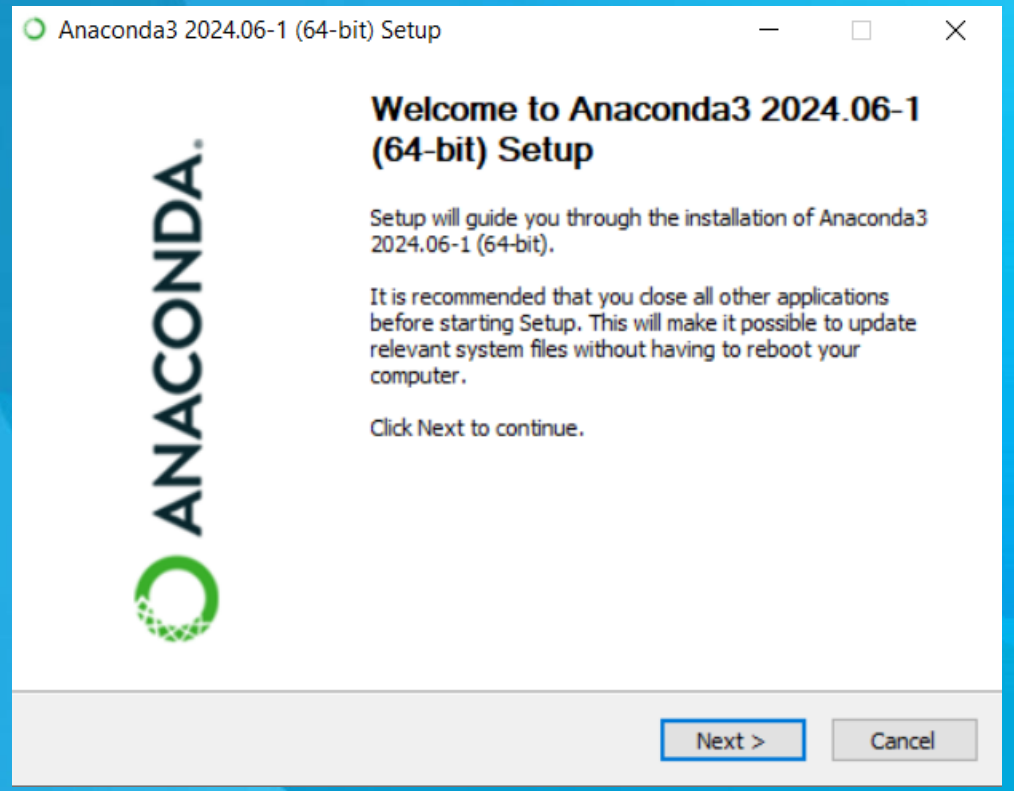


2- Press on Free Download

3- The environment will start downloading, when finish you will have a file look like this:



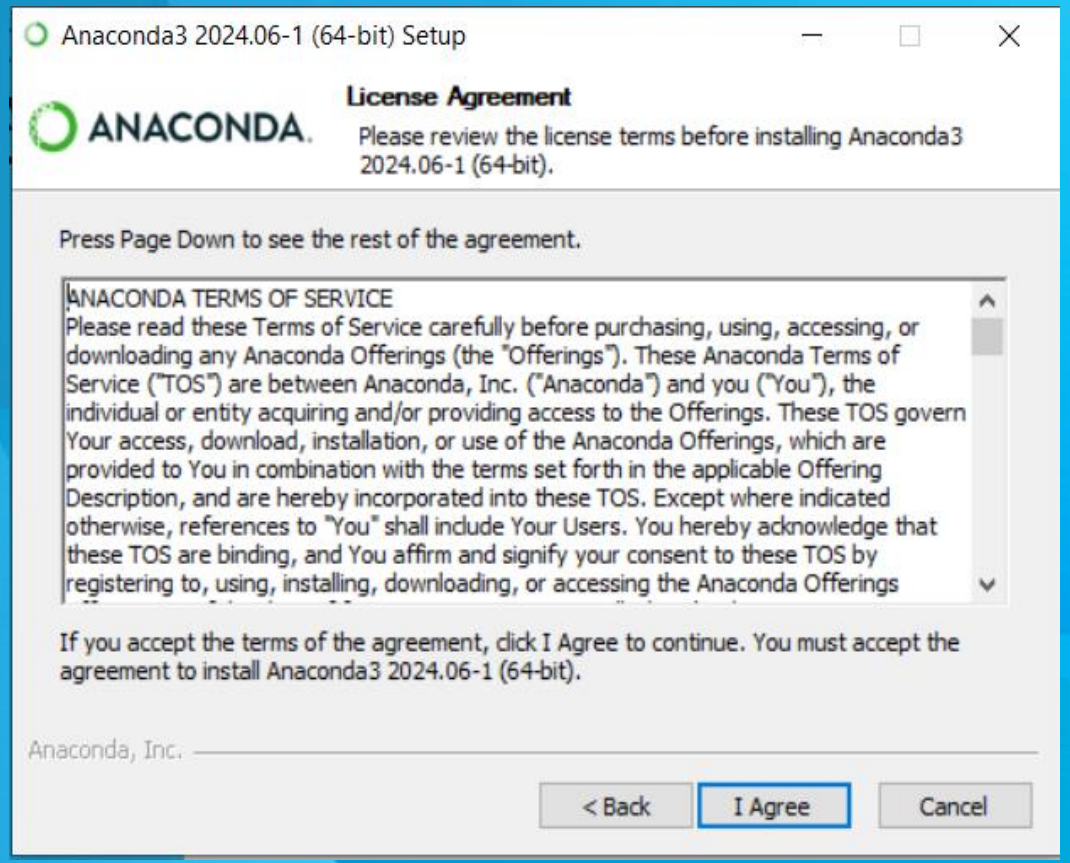
Python Programming Languages:
Preparing of Python Environment:
The Best Environment for Python called Anaconda



Python Programming Languages:

Preparing of Python Environment:

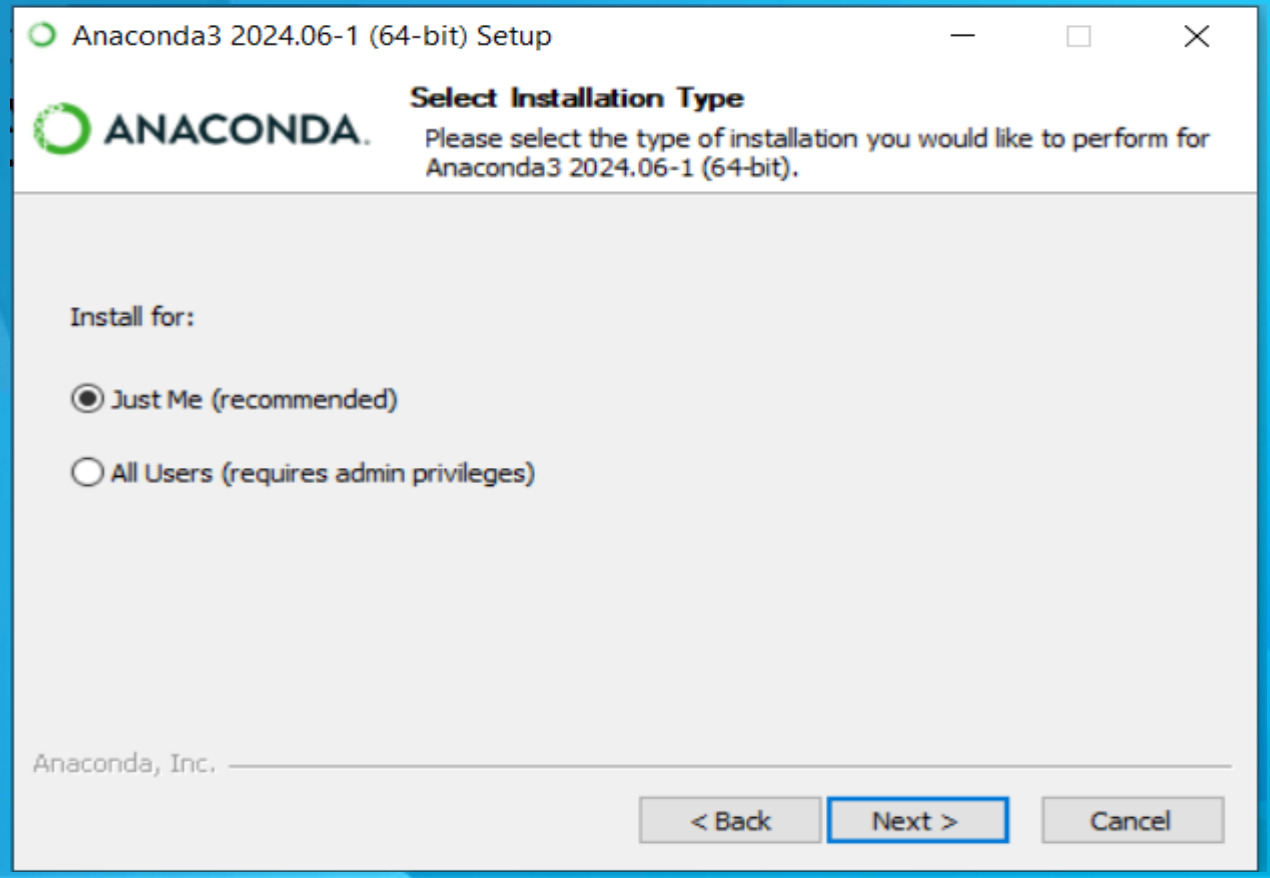
The Best Environment for Python called Anaconda



Python Programming Languages:

Preparing of Python Environment:

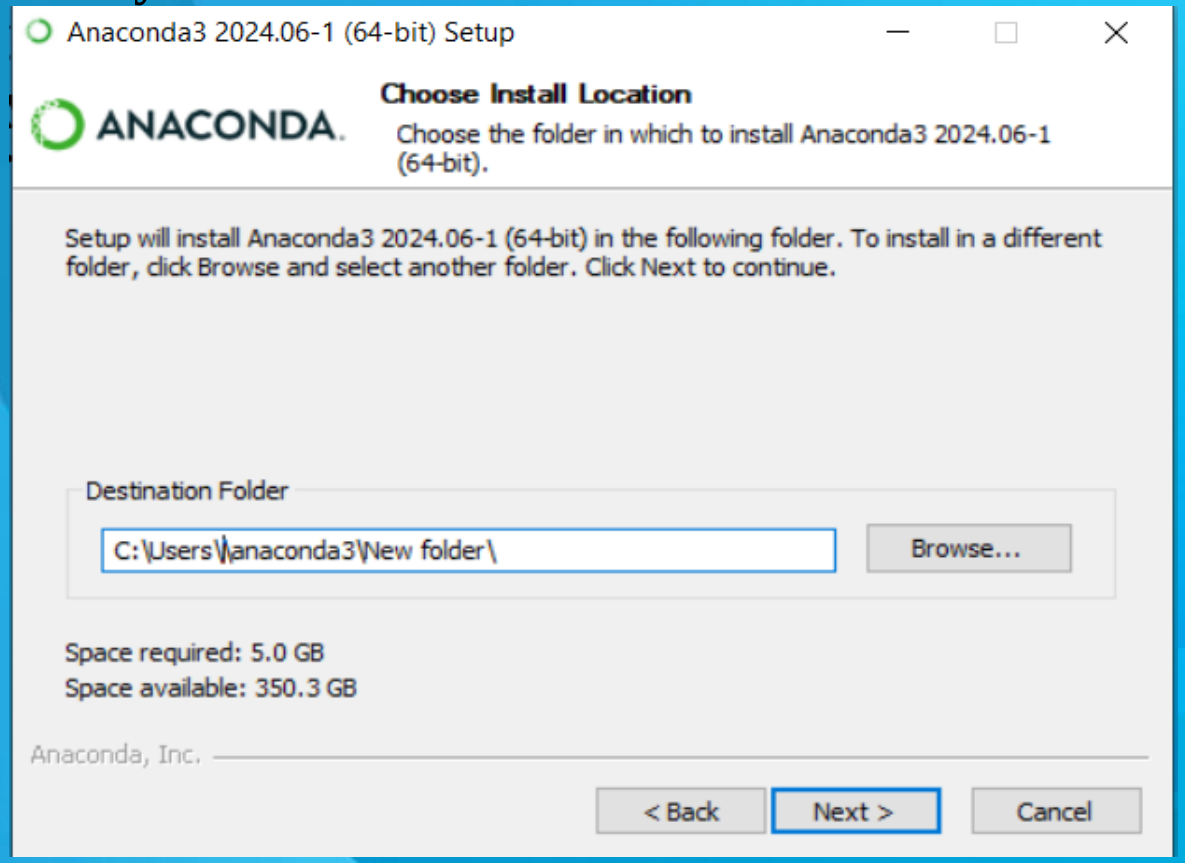
The Best Environment for Python called Anaconda



Python Programming Languages:

Preparing of Python Environment:

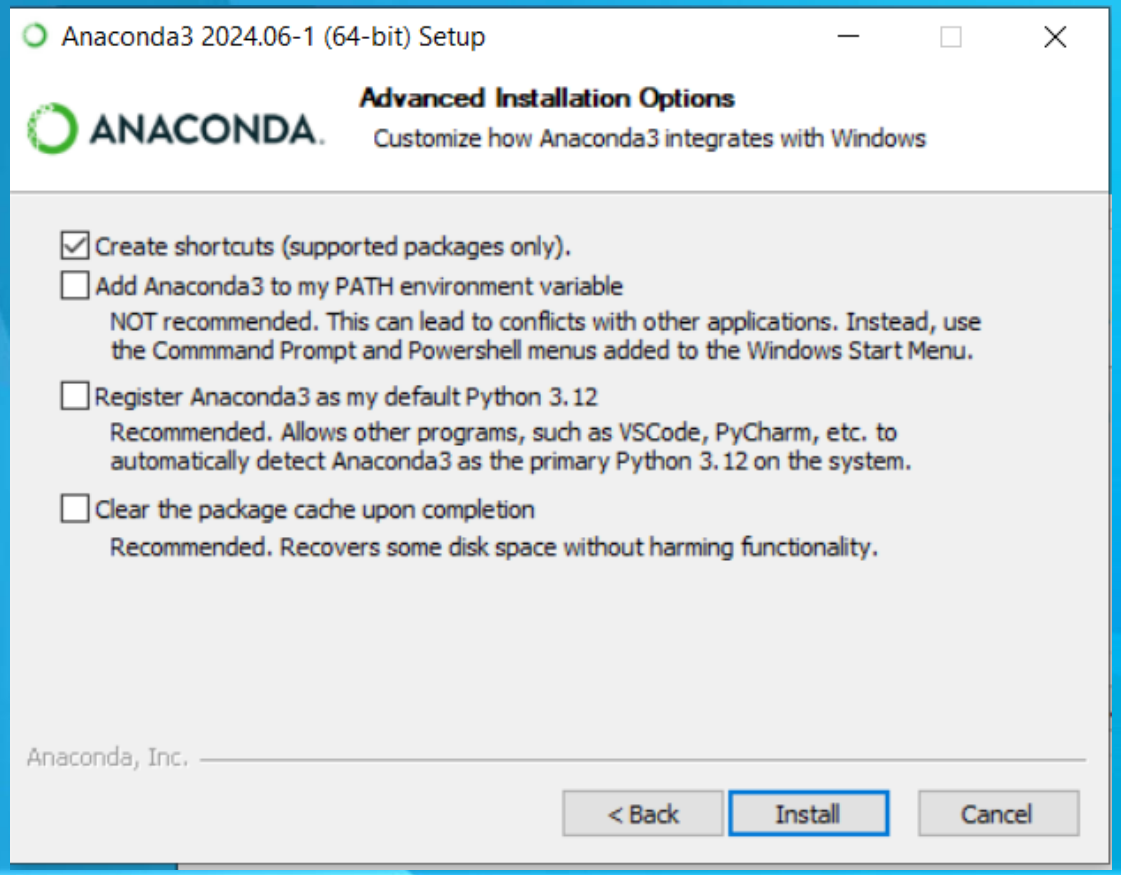
The Best Environment for Python called Anaconda



Python Programming Languages:

Preparing of Python Environment:

The Best Environment for Python called Anaconda

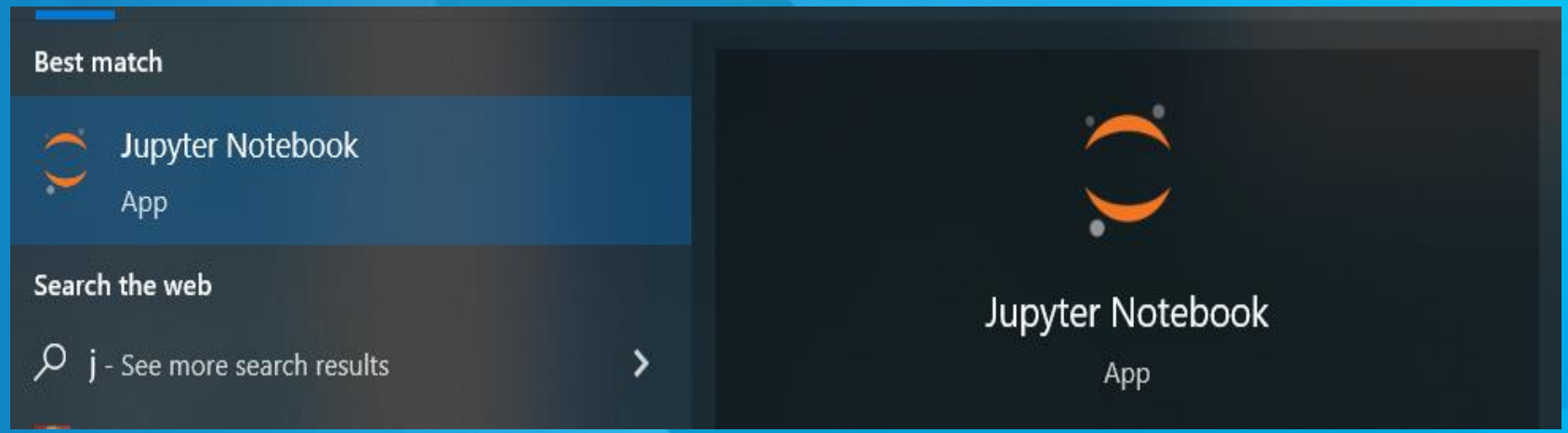


Python Programming Languages:

Preparing of Python Environment:

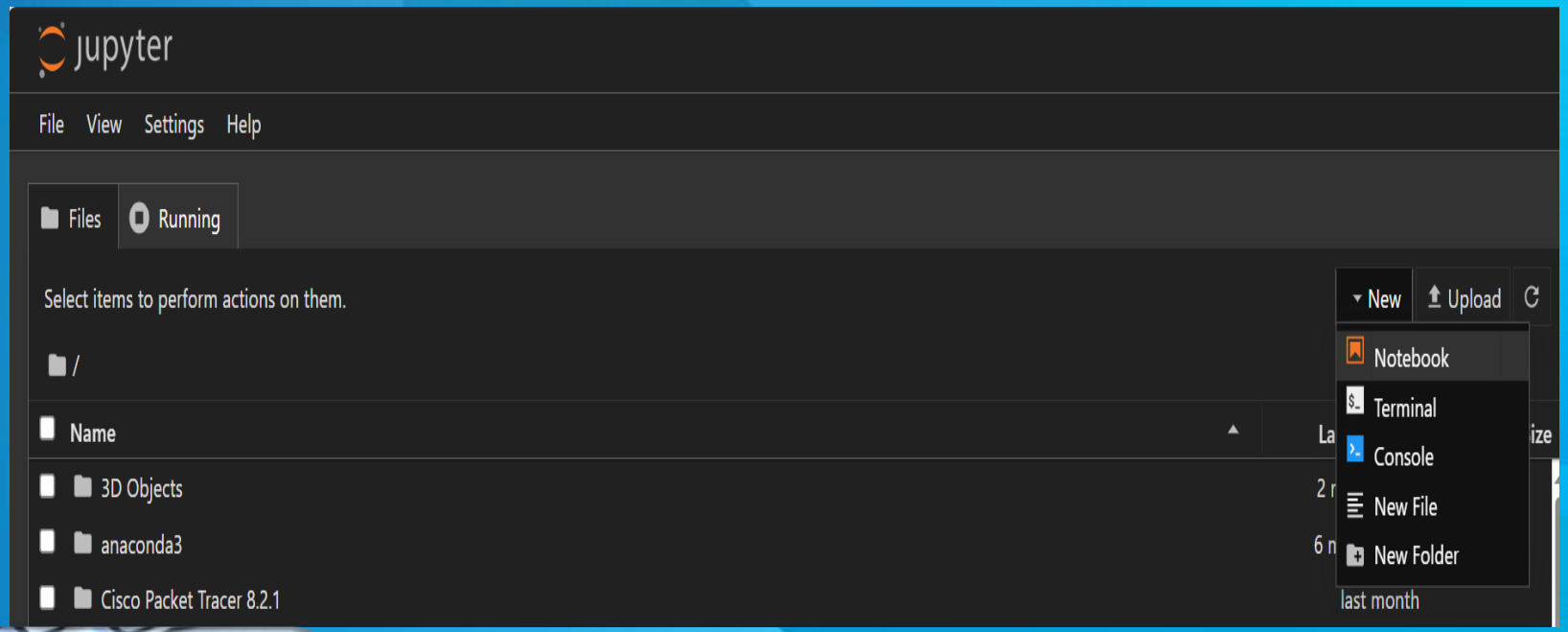
The Best Environment for Python called Anaconda

1- From start menu open and type Jupyter.



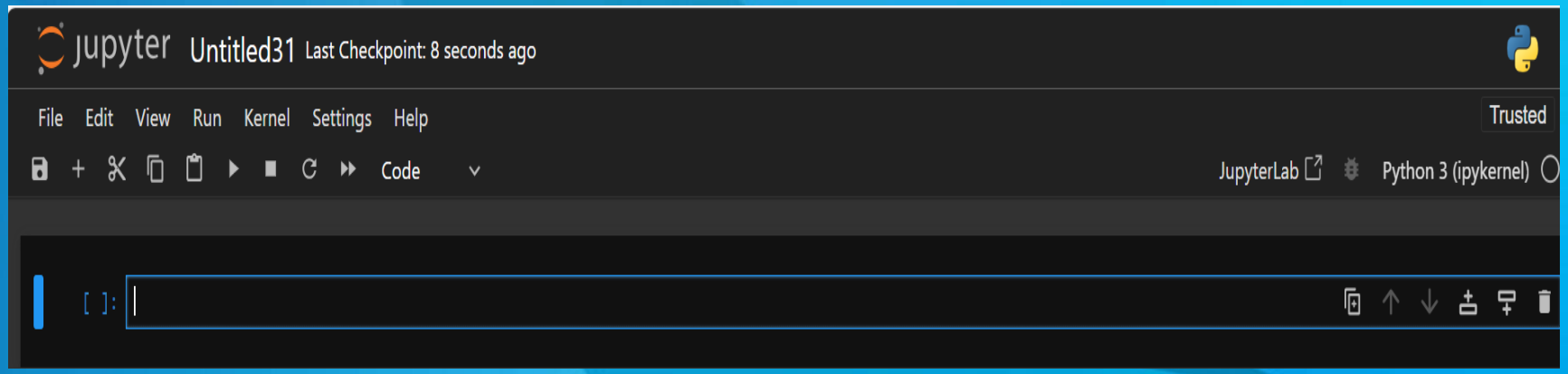
Python Programming Languages:
Preparing of Python Environment:
The Best Environment for Python called Anaconda

2- The browser now will open automatically



Python Programming Languages:
Preparing of Python Environment:
The Best Environment for Python called Anaconda

3- Jupyter interface:



Python Programming Languages:

Preparing of Python Environment:

The Best Environment for Python called Anaconda

4- If your PC or Laptop is old and with low resources or you don’t have Laptop, don’t worry you can work with Google Colab:

https://colab.research.google.com/

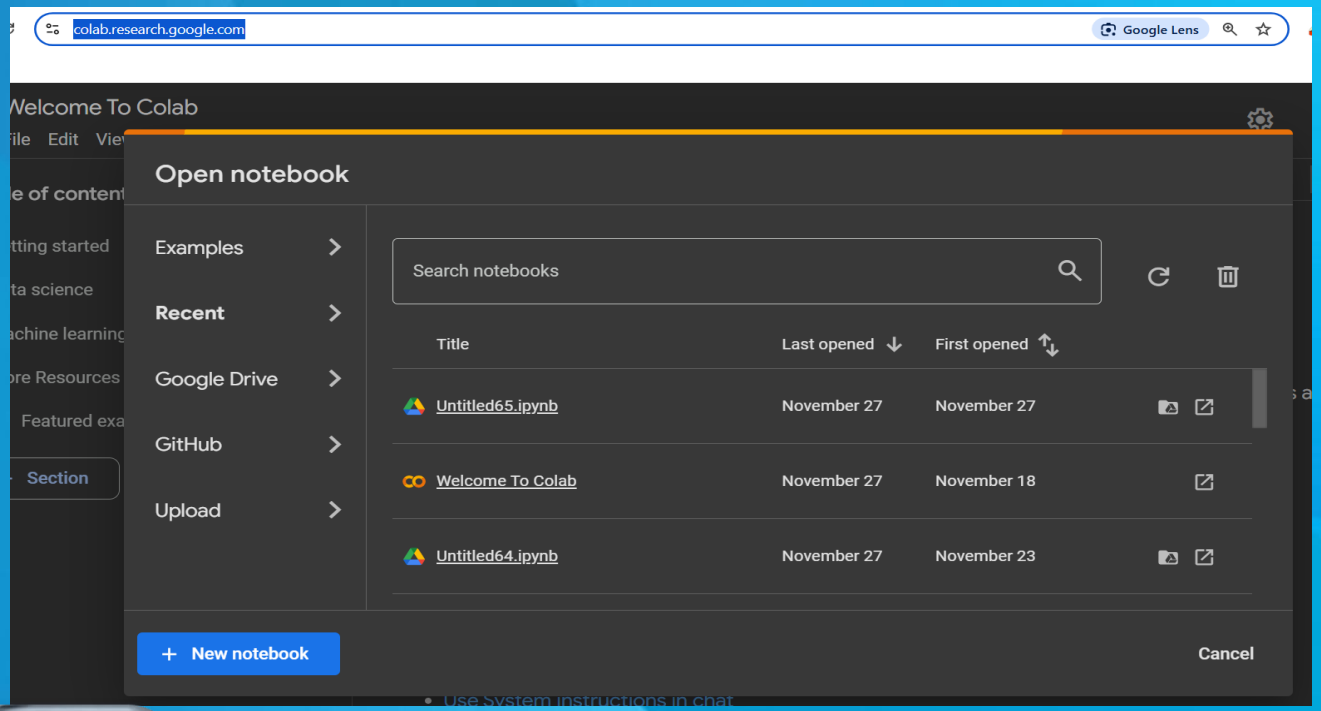
It’s a free platform from google that allow you to run any Python codes and you also can create and manage a full project in it.

Python Programming Languages:

Preparing of Python Environment:

The Best Environment for Python called Anaconda

During this course, we will focus our work on Google Colab

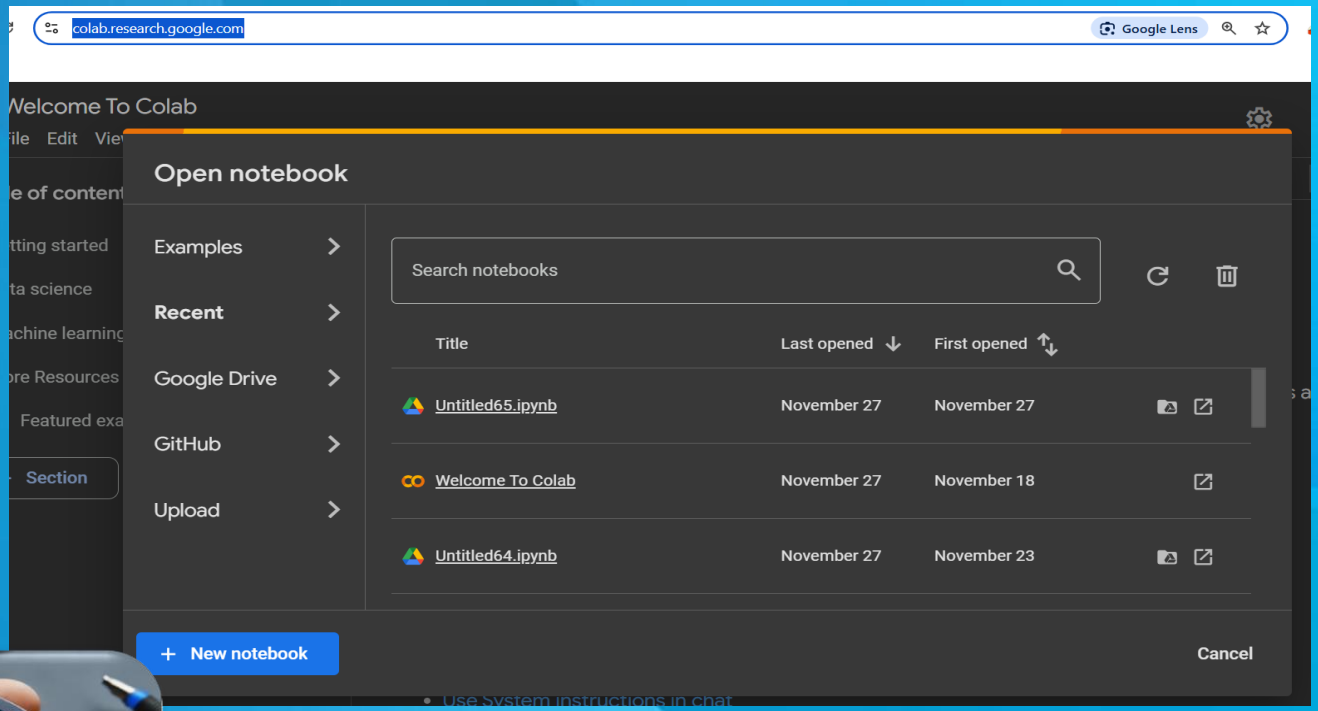


Python Programming Languages:

Preparing of Python Environment:

The Best Environment for Python called Anaconda

4- If your PC or Laptop is old and with low resources or you don't have Laptop, don't worry you can work with Google Colab:



Python Programming Languages:

Lets dive into Python, from Scratch!

Variables

Think of a **variable** as a container that stores information. Just like you might put a name label on a box to remember what's inside, a variable helps your program remember a value.

•**Example:**

If you want to store your age, you can use a variable like age = 25.
In Python, creating a variable is as simple as giving it a name and assigning it a value using the = symbol.

Python Programming Languages:

Lets dive into Python, from Scratch!

Rules for Variable Names:

1 - **Must** start with a letter (a-z, A-Z) or an underscore (_)
Example: name , _count

2- **Cannot** start with a number.
Name : Correct, 1name: Wrong.

3- **No** spaces or special characters:
@#$%: don’t use , firstname: correct, first_name: correct, first name: Wrong

4- Variables are **case-sensitive**: Name is different from name. A is totally different from a.

Python Programming Languages:

Lets dive into Python, from Scratch!

Numbers in Python

1- integers: 1, -5 , +800 , …

2- float: 1.5 , -0.8, -3.1486458.. , +18.26…

Basic Operations with Numbers:

Operation	Symbol	Example	Result
Addition	`+`	`5 + 3`	`8`
Subtraction	`-`	`10 - 4`	`6`
Multiplication	`*`	`6 * 7`	`42`
Division	`/`	`15 / 3`	`5.0`
Modulus (remainder)	`%`	`10 % 3`	`1`
Exponent (power)	`**`	`2 ** 3`	`8`

Python Programming Languages:

Lets dive into Python, from Scratch!

Strings in Python:

String is a sequence of characters—letters, numbers, symbols—enclosed in quotation marks.

- **Single quotes: ‘hello’ or Double quotes: “hello” are the same in Python.**
- **To handle a string consist of multiple sentences and lines we use triple quotes.**

“ ” ” Welcome to Python programming language. It widely used in Artificial Intelligence and Bioinformatics” ” ”

String Operations:

Operation	Example	Result
Concatenation (joining)	`'Hello' + ' World'`	`'Hello World'`
Repetition	`'Hi' * 3`	`'HiHiHi'`
Length	`len('Python')`	`6`

Python Programming Languages:

Lets dive into Python, from Scratch!

Strings in Python:

String is a sequence of characters—letters, numbers, symbols—enclosed in quotation marks.

- Accessing Characters in a String:

You can access individual characters in a string using **indexing** (position of characters).

Example:

```
word = "Python"
first_letter = word[0]  # 'P' (index starts at 0)
last_letter = word[-1]  # 'n' (negative index starts from the end)
```

Python Programming Languages:

Lets dive into Python, from Scratch!

Booleans in Python:

Boolean represents one of two values: True , False

Boolean Operations:

Operation	Symbol	Example	Result
Equal to	`==`	`5 == 5`	`True`
Not equal to	`!=`	`5 != 3`	`True`
Greater than	`>`	`7 > 3`	`True`
Less than	`<`	`2 < 5`	`True`
Greater than or equal to	`>=`	`6 >= 6`	`True`
Less than or equal to	`<=`	`4 <= 8`	`True`

Python Programming Languages:

Lets dive into Python, from Scratch!

Now that we've covered the basics of variables, numbers, strings, and Booleans, it's time to introduce one of the most important and versatile data structures in Python: **Lists**.

Think of a **list** as a **collection of items**. Just like a shopping list holds multiple items, a Python list can hold multiple values—numbers, strings, or even other lists!

Python Programming Languages:

Lets dive into Python, from Scratch!

What is a List?

A **list** is a collection of items (or elements) that are ordered and changeable (mutable). Lists are defined using **square brackets [ ] ,** and the elements inside the list are separated by commas.

Example:

Fruits = [‘apple’ , ‘banana’, ‘cherry’]

Here Fruits represent a variable which is a list containing three strings elements : ‘apple’ , ‘banana’, ‘cherry’

Python Programming Languages:

Lets dive into Python, from Scratch!

What is a List?

A **list** is a collection of items (or elements) that are ordered and changeable (mutable). Lists are defined using **square brackets [ ] ,** and the elements inside the list are separated by commas.

Example:

List of numbers:

Numbers = [1 , 2, 3 , 4 , 5]

List of strings:

Names = [‘Ahmad’ , ‘Sara’ , ‘Lama’ , ‘Mohammad’]

Mixed list:

Mixed = [1 , ‘hello’ , True, 3.148]

Python Programming Languages:

Lets dive into Python, from Scratch!

What is a List?

Accessing Elements in a List:

You can access individual elements of a list using **indexing**.

- **Indexing starts from 0** (just like strings).
- **Negative indexing** starts from the end.

Example:

```
fruits = ['apple', 'banana', 'cherry']
print(fruits[0])    # Output: 'apple'
print(fruits[1])    # Output: 'banana'
print(fruits[-1])   # Output: 'cherry' (last element)
```

Python Programming Languages:

Lets dive into Python, from Scratch!

What is a List?

Changing Elements in a List

Lists are **mutable**, meaning you can change their content after they are created.

Example:

```
fruits = ['apple', 'banana', 'cherry']
fruits[1] = 'blueberry'  # Change 'banana' to 'blueberry'
print(fruits)  # Output: ['apple', 'blueberry', 'cherry']
```

Basic List Operations

Operation	Description	Example	Result
Length of list	`len()` returns the number of items	`len([1, 2, 3])`	`3`
Append (add to end)	`list.append(item)` adds an item	`fruits.append('orange')`	`['apple', 'banana', 'cherry', 'orange']`
Insert (add at index)	`list.insert(index, item)` adds an item at a specific index	`fruits.insert(1, 'grape')`	`['apple', 'grape', 'banana', 'cherry']`
Remove by value	`list.remove(item)` removes an item	`fruits.remove('banana')`	`['apple', 'cherry']`
Pop (remove by index)	`list.pop(index)` removes an item at a specific index (default is last item)	`fruits.pop()`	`['apple', 'banana']` (removes 'cherry')
Concatenation	`+` joins two lists	`[1, 2] + [3, 4]`	`[1, 2, 3, 4]`
Repetition	`*` repeats a list	`[1, 2] * 3`	`[1, 2, 1, 2, 1, 2]`

Python Programming Languages:

Lets dive into Python, from Scratch!

What is a List?

Slicing Lists:

You can extract a portion of a list using **slicing**.

Syntax: list[start : end]

Start from the **start** index and go up to **end** without including it

Example:

```
numbers = [0, 1, 2, 3, 4, 5]
slice1 = numbers[1:4]  # From index 1 to 3
print(slice1)  # Output: [1, 2, 3]
```

Python Programming Languages:

Lets dive into Python, from Scratch!

What is a List?

Slicing Lists:

You can also use **step** in slicing:

```
numbers = [0, 1, 2, 3, 4, 5]
slice2 = numbers[0:6:2]  # Every second element
print(slice2)  # Output: [0, 2, 4]
```

Python Programming Languages:

Lets dive into Python, from Scratch!

What is a List?

Looping Through a List:

You can use a **for loop** to iterate through each item in a list.

Example:

```
fruits = ['apple', 'banana', 'cherry']
for fruit in fruits:
    print(fruit)
```

List Methods

Here are some commonly used methods in Python lists:

Method	Description	Example
`append(item)`	Adds an item to the end of the list	`fruits.append('orange')`
`insert(index, item)`	Inserts an item at a specific index	`fruits.insert(1, 'grape')`
`remove(item)`	Removes the first occurrence of an item	`fruits.remove('apple')`
`pop(index)`	Removes and returns an item at an index	`fruits.pop(0)`
`index(item)`	Returns the index of the first occurrence	`fruits.index('banana')`
`count(item)`	Counts how many times an item appears	`fruits.count('apple')`
`sort()`	Sorts the list in ascending order	`numbers.sort()`
`reverse()`	Reverses the order of the list	`numbers.reverse()`

Python Programming Languages:

Lets dive into Python, from Scratch!

What is a List?

Nested Lists: this is how we will handle Biological Data!

A **nested list** is a list inside another list.

Example:

```
nested_list = [[1, 2, 3], ['a', 'b', 'c']]
print(nested_list[0])  # Output: [1, 2, 3]
print(nested_list[0][1])  # Output: 2
```

Python Programming Languages:

Lets dive into Python, from Scratch!

Now that we've explored lists, let's move on to another essential data structure in Python: **Dictionaries**.

A **dictionary** is like a real-world dictionary where you look up a word (the key) to find its meaning (the value). In Python, dictionaries are used to store data in **key-value pairs**. They are powerful, flexible, and widely used in programming.

Python Programming Languages:

Lets dive into Python, from Scratch!

What is a Dictionary?

A **dictionary** is a collection of key-value pairs, where each key is unique, and it is used to access its corresponding value.
Keys can be strings, numbers, or even lists.
Values can be any data type, including numbers, strings, lists, or even other dictionaries.
Creating a Dictionary:

```
student = {
    'name': 'Alice',
    'age': 22,
    'major': 'Biology'
}
```

name is the 1st key, Alice is its value
age is the 2nd key, 22 is its value
Major is the 3rd key, Biology is its value

Python Programming Languages:

Lets dive into Python, from Scratch!

What is a Dictionary?

Accessing Values in a Dictionary:

To access a Value, you can use the key inside square brackets [ ] or the get( ) method.

Example:

```
student = {'name': 'Alice', 'age': 22, 'major': 'Biology'}
print(student['name'])  # Output: Alice
print(student.get('age'))  # Output: 22.
```

Adding or Updating Key-Value Pairs:

You can add a new key-value pair or update an existing one by assigning a value to a key.

```
student = {'name': 'Alice', 'age': 22}
student['major'] = 'Biology'  # Adding a new key-value pair
student['age'] = 23  # Updating the value of an existing key
print(student)  # Output: {'name': 'Alice', 'age': 23, 'major': 'Biology'}
```

Python Programming Languages:

Lets dive into Python, from Scratch!

What is a Dictionary?

Removing Key-Value Pairs:

You can remove a key-value pair using the del statement or the pop( ) method.

Example:

```
student = {'name': 'Alice', 'age': 22, 'major': 'Biology'}
del student['major']  # Removes the key 'major'
print(student)  # Output: {'name': 'Alice', 'age': 22}
age = student.pop('age')  # Removes and returns the value of 'age'
print(student)  # Output: {'name': 'Alice'}
print(age)  # Output: 22
```

Python Programming Languages:

Lets dive into Python, from Scratch!

What is a Dictionary?

Looping Through a Dictionary

You can loop through a dictionary to access its keys, values, or both.

Example:

```
student = {'name': 'Alice', 'age': 22, 'major': 'Biology'}
# Loop through keys
for key in student:
    print(key)  # Output: name, age, major

# Loop through values
for value in student.values():
    print(value)  # Output: Alice, 22, Biology

# Loop through key-value pairs
for key, value in student.items():
    print(key, ':', value)
```

Python Programming Languages:

Lets dive into Python, from Scratch!

What is a Dictionary?

Nested Dictionaries:

You can store dictionaries inside other dictionaries, which is useful for representing more complex data.

```
students = {
    'student1': {'name': 'Alice', 'age': 22},
    'student2': {'name': 'Bob', 'age': 24}
}
print(students['student1']['name'])  # Output: Alice
```

Python Programming Languages:

Dictionary Methods

Here are some commonly used dictionary methods:

Method	Description	Example
`dict.get(key)`	Returns the value for the specified key	`student.get('name')`
`dict.keys()`	Returns all keys as a list	`student.keys()`
`dict.values()`	Returns all values as a list	`student.values()`
`dict.items()`	Returns all key-value pairs	`student.items()`
`dict.update()`	Updates the dictionary with key-value pairs from another dictionary	`student.update({'grade': 'A'})`
`dict.clear()`	Removes all items from the dictionary	`student.clear()`

Python Programming Languages:

Basic Dictionary Operations

Operation	Description	Example	Result
Access a value	`dict[key]`	`student['name']`	`'Alice'`
Add/Update a value	`dict[key] = value`	`student['age'] = 23`	`{'name': 'Alice', 'age': 23}`
Remove a key-value pair	`del dict[key]` or `dict.pop(key)`	`del student['age']`	`{'name': 'Alice'}`
Check if key exists	`key in dict`	`'age' in student`	`True`
Get all keys	`dict.keys()`	`student.keys()`	`dict_keys(['name', 'age'])`
Get all values	`dict.values()`	`student.values()`	`dict_values(['Alice', 22])`
Get all key-value pairs	`dict.items()`	`student.items()`	`dict_items([('name', 'Alice'), ('age', 22)])`