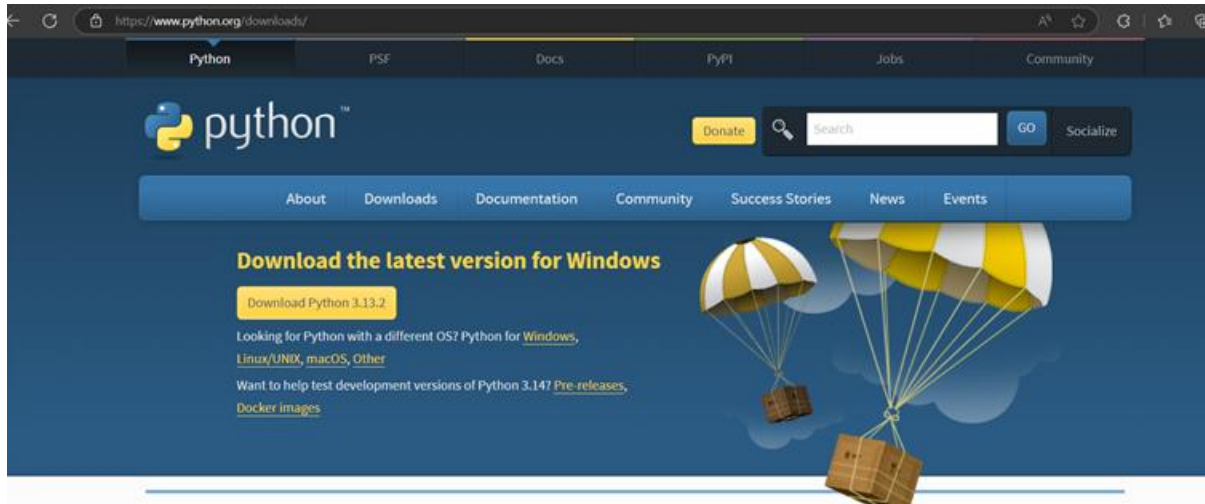


## Day 1: How to download Python?

Go to google and search for download python



Click on the download python button. After downloading install the python

After installing we have set a path so,

Open file manager and go to C drive and follow this path.

**C:\Users\{username}\AppData\Local\Programs\Python\Python313**

**C:\Users\{username}\AppData\Local\Programs\Python\Python313\Scripts**

Copy and search for Edit the system environment variable -> Environment Variables -> on system variable double click on path -> then paste the path here that you copy.

Then click ok -> ok -> and apply.

Now go to the cmd and check the python is downloaded or not in our system.

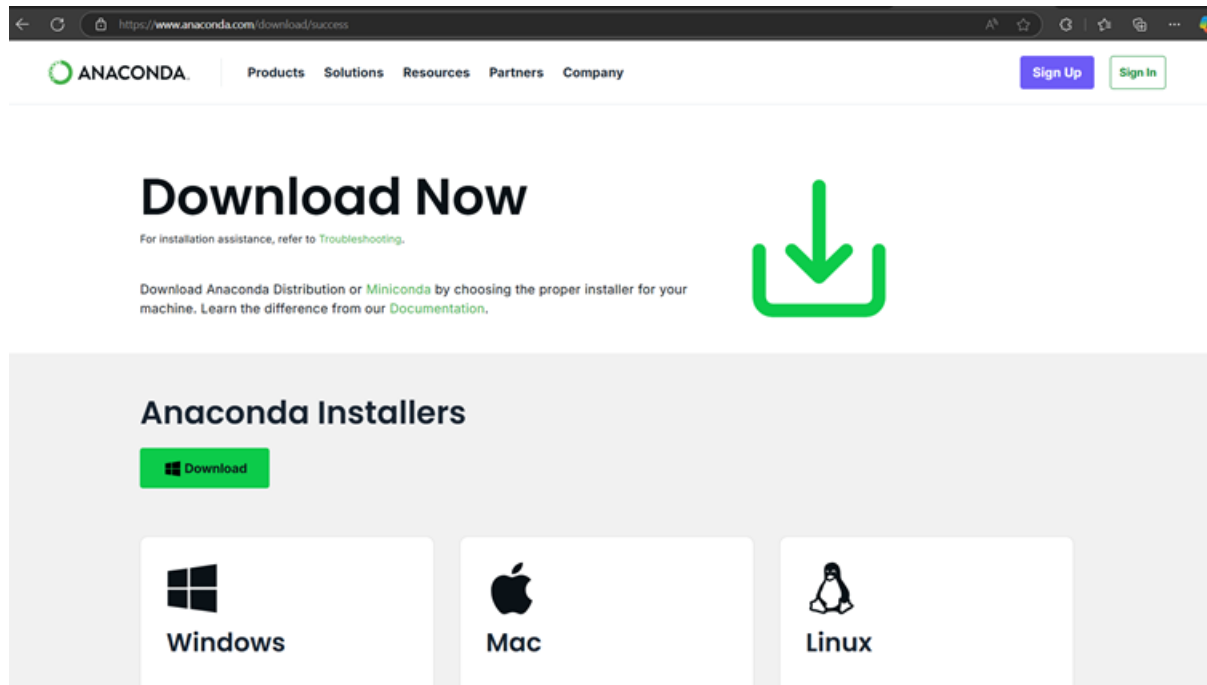
```
Microsoft Windows [Version 10.0.26100.3194]
(c) Microsoft Corporation. All rights reserved.

C:\Users\Amol Thakare>python --version
Python 3.13.1

C:\Users\Amol Thakare>
```

Now I used Jupiter notebook for my learning

So, here is how it downloads. First go to google and search for Download anaconda



Then click on the Download button. According to your system it directly downloads the anaconda.

After downloading, install the anaconda.

I face one issue while accessing the jupyter notebook.

First, we must check that the Jupiter notebook is install or not so go to cmd and type Jupiter notebook

```
jupyter' is not recognized as an internal or external command,  
perable program or batch file.
```

I got this error.

So, if you encounter this error then simply go this path

**C:\Users\{username}\anaconda3\Scripts**

**C:\Users\{username}\anaconda3\Library\bin**

And same as for setting path for python. Do the same thing for this also.

## Day 2: Python Comments

Comments are hints that we add to our code to make it easier to understand.

There are two types of comments:

- 1) **Single-line comment**
- 2) **Multiline comment**

### Single-line Comment:

We use the hash (#) symbol to write a single-line comment. For example,

```
# print a number
```

```
print(25) # number
```

A single-line comment starts with # and extends up to the end of the line. We can also use single-line comments alongside the code:

### Multiline Comments:

Unlike languages such as C++ and Java, Python doesn't have a dedicated method to write multi-line comments.

“we use Multiline comments in the next example”

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

### Why Use Comments?

We should use comments:

For future references, as comments make our code readable.

For debugging.

For code collaboration, comments help peer developers to understand each other's code.

### Python Type Conversion:

Python conversion typically refers to converting one data type to another. This can include changing a string to a number, a list to a set, and so on. Here are some common conversions with examples:

#### 1. String to Integer

You can convert a string to an integer using the `int()` function.

```
string_value = "123"  
integer_value = int(string_value)  
print(integer_value) # Output: 123
```

```
print(type(integer_value)) # Output: <class 'int'>
```

## 2. Integer to String

Convert an integer to a string using the `str()` function.

```
integer_value = 123
string_value = str(integer_value)
print(string_value) # Output: "123"
print(type(string_value)) # Output: <class 'str'>
```

## 3. Float to Integer

Use the `int()` function to convert a float to an integer (note that this truncates the decimal part).

```
float_value = 123.45
integer_value = int(float_value)
print(integer_value) # Output: 123
```

## 4. Integer to Float

Convert an integer to a float using the `float()` function.

```
integer_value = 123
float_value = float(integer_value)
print(float_value) # Output: 123.0
```

## 5. List to Set

You can convert a list to a set using the `set()` function.

```
list_value = [1, 2, 3, 3, 4]
set_value = set(list_value)
print(set_value) # Output: {1, 2, 3, 4}
```

## 6. Set to List

Convert a set to a list using the `list()` function.

```
set_value = {1, 2, 3, 4}
list_value = list(set_value)
print(list_value) # Output: [1, 2, 3, 4]
```

## 7. Dictionary to List

Extract keys or values from a dictionary to form a list.

```
dict_value = {"a": 1, "b": 2, "c": 3}
list_keys = list(dict_value.keys())
list_values = list(dict_value.values())
print(list_keys) # Output: ['a', 'b', 'c']
print(list_values) # Output: [1, 2, 3]
```

### Key Points to Remember:

- 1) Type Conversion is the conversion of an object from one data type to another data type.
- 2) Implicit Type Conversion is automatically performed by the Python interpreter.
- 3) Python avoids the loss of data in Implicit Type Conversion.
- 4) Explicit Type Conversion is also called Type Casting, the data types of objects are converted using predefined functions by the user.
- 5) In Type Casting, loss of data may occur as we enforce the object to a specific data type.

## Python Basic Input and Output:

In Python, basic input and output (I/O) operations are essential for interacting with the user or other systems. Here are some fundamental ways to handle input and output in Python,

### Input

You can get input from the user using the `input()` function. The `input()` function reads a line from input, converts it to a string (if needed), and returns it.

Example: Getting User Input

**# Prompt the user for their name**

```
name = input("Enter your name: ")
```

**# Prompt the user for their age**

```
age = input("Enter your age: ")
```

**# Print a message using the input values**

```
print("Hello, " + name + "! You are " + age + " years old.")
```

### Output

You can display output using the `print()` function. The `print()` function writes the value to the standard output (usually the console).

Example: Printing Output

**# Display a simple message**

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

**# Display multiple items**

```
name = "Alice"
```

```
age = 30
```

```
print("Name:", name, "Age:", age)
```

**# Using formatted strings for output**

```
print(f"Hello, {name}! You are {age} years old.")
```

## Handling Different Data Types

If you need to handle different data types, you may need to convert them appropriately.

Example: Converting Input

**# Prompt the user for their age and convert it to an integer**

```
age = int(input("Enter your age: "))
```

**# Perform some calculation**

```
years_until_100 = 100 - age
```

**# Print the result**

```
print(f"You will turn 100 years old in {years_until_100} years.")
```

## Combining Input and Output

Combining both input and output operations allows for interactive programs.

Example: Simple Interactive Program

**# Prompt the user for their name**

```
name = input("What's your name? ")
```

**# Greet the user**

```
print(f"Nice to meet you, {name}!")
```

**# Ask for their favorite color**

```
color = input("What's your favorite color? ")
```

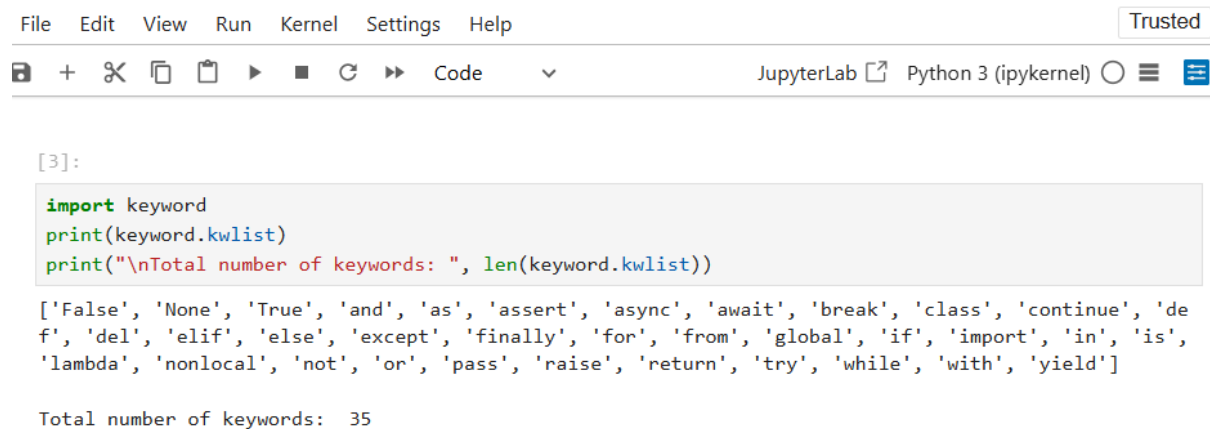
**# Respond to their input**

```
print(f"{color} is a great color, {name}!")
```

## Day 3: Python Keywords

**Keywords** in Python are reserved words that have specific meanings and purposes in the language. They are used to define the syntax and structure of Python programs. You cannot use keywords such as variable names, function names, or any other identifiers. Python has a list of keywords that you can view by running the following code:

```
import keyword
print(keyword.kwlist)
```



The screenshot shows a JupyterLab window with a menu bar (File, Edit, View, Run, Kernel, Settings, Help) and a toolbar. The code cell contains the following Python code:

```
[3]:
import keyword
print(keyword.kwlist)
print("\nTotal number of keywords: ", len(keyword.kwlist))
```

The output of the code is displayed below the cell:

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

Total number of keywords:  35
```

### Example

```
if True:
    print("This is a keyword example")
```

### Identifiers:

**Identifiers** are names given to variables, functions, classes, and other objects in Python. Identifiers can be a combination of letters (a-z, A-Z), digits (0-9), and underscore (\_). They must start with a letter or an underscore and cannot contain spaces or special characters.

#### Rules for Identifiers:

1. Must start with a letter or underscore.
2. Can contain letters, digits, and underscores.
3. Case-sensitive (MyVar and myvar are different identifiers).
4. Cannot be a keyword.

#### Example

```
my_variable = 10
MyVariable = 20
print(my_variable, MyVariable) #Output: 10 20
```

[5]:

```
abc_12 = 12
print(abc_12)
```

12

[6]:

```
global = 1
```

Cell In[6], line 1

```
global = 1
```

^

SyntaxError: invalid syntax

[7]:

```
a@ = 10
```

Cell In[7], line 1

```
a@ = 10
```

^

SyntaxError: invalid syntax

we cannot use special symbols like !, @, #, \$, % etc. in our identifier.

## Indentation:

**Indentation** in Python is crucial as it indicates a block of code. Unlike many other programming languages that use curly braces { } to define code blocks, Python uses indentation (whitespace) to group statements. Consistent indentation is required for defining loops, functions, classes, conditionals, and other structures.

## Rules for Indentation:

1. Use the same number of spaces (typically 4 spaces) for each level of indentation.
2. Indentation should be consistent throughout the code block.

## Example

```
# Correct indentation
```

```
for i in range(5):
```

```
    print(i) #Indented block
```

```
    if i % 2 == 0:
```

```
        print(f"{i} is even") #Nested indented block
```

```
# Incorrect indentation will raise an error
```

```
for i in range(5):
```

```
print(i) #IndentationError: expected an indented block
```



## Day 4: Python Variables and Data Types

### Variables:

Variables in Python are used to store data. A variable name can contain letters, digits, and underscores, but it must start with a letter or an underscore. For example:

```
# Valid variable names
name = "Alice"
age = 25
_height = 5.9
```

### Data Types:

Every value in Python has a datatype. Since everything is an object in Python programming, data types are actually classes and variables are instance(object) of these classes.

### Numbers:

Integers, floating point numbers and complex numbers fall under Python numbers category. They are defined as int, float and complex classes in Python.

We can use the type() function to know which class a variable or a value belongs to and the isinstance() function to check if an object belongs to a particular class.

### Integers (int):

- Whole numbers, positive or negative.

```
a = 5
print(a, "is of type", type(a))
```

```
5 is of type <class 'int'>
```

### Floating Point (float):

- Numbers with a decimal point.

```
a = 5.5
print(a, "is of type", type(a))
```

```
5.5 is of type <class 'float'>
```

## String (str):

- A sequence of characters, surrounded by single or double quotes.

```
a = "Hello"  
print(a, "is of type", type(a))
```

Hello is of type <class 'str'>

## Boolean (bool):

- Represents True or False values.

```
a = True  
print(a, "is of type", type(a))
```

True is of type <class 'bool'>

## List:

- An ordered collection of items, which can be of different data types.
- list is an ordered of items. It is one of the most used datatype in python and is very flexible. All the items in a list do not need to be of the same type.
- Declaring a list is, Items separated by commas are enclosed within brackets [ ].

```
a = [1,2,3,4]  
print(a, "is of type", type(a))
```

[1, 2, 3, 4] is of type <class 'list'>

```
a = [10,11,20.3,"Hello"]  
print(a[2])
```

20.3

Lists are mutable means values of the elements of a list can be altered

```
a[1] = 45.7  
print(a)
```

[10, 45.7, 20.3, 'Hello']

## Tuple:

- Tuples is an ordered of items same as list. The only difference is that tuples are immutable. Tuples once created cannot be modified.

```
t = (1, 4.34, "HFHG")  
print(t)
```

```
(1, 4.34, 'HFHG')
```

```
t[1] = 2.3
```

```
-----  
TypeError                                Traceback (most recent  
Cell In[10], line 1  
----> 1 t[1] = 2.3  
  
TypeError: 'tuple' object does not support item assignment
```

## Set:

- An unordered collection of unique items.
- set is defined by values separated by comma inside braces{}. Items in a set are not ordered.

```
a = {1,2,3,4}  
print(a, "is of type", type(a))
```

```
{1, 2, 3, 4} is of type <class 'set'>
```

We cannot perform set operations like union and intersection on two sets. Sets have unique values.

```
a = {45, 67, 80, 56, 45, 80}  
print(a)
```

```
{80, 56, 67, 45}
```

```
print(a[1])
```

```
-----  
TypeError                                Traceback (most recent call last)  
Cell In[18], line 1  
----> 1 print(a[1])  
  
TypeError: 'set' object is not subscriptable
```

We cannot print a particular element in the set because it is an unordered collection of items.

## Dictionary (dict):

- A collection of key-value pairs.
- In python, Dict are defined within braces {} with each item being pair in the form of key:value. Key and value can be of any type.

```
d = {'a': "apple", 'b': "bat"}  
print(d, "is of type", type(d))
```

```
{'a': 'apple', 'b': 'bat'} is of type <class 'dict'>
```

## Day 5: Operators

Operators are special in Python that carry out arithmetic or logical computation. The value that the operator operates is called operand.

### Operator Types:

1. Arithmetic Operator
2. Comparison Operator
3. Logical Operator
4. Bitwise Operator
5. Assignment Operator
6. Special Operator

### Arithmetic Operator:

These operators are used to perform basic mathematical operations:

```
a = 3
b = 5
print(a+b) #Addition
print(a-b) #Subtraction
print(a*b) #Multiplication
print(a/b) #Division
print(a//b) #Floor Division (division that rounds down to the nearest whole number)
print(a**b) #Exponentiation (power)
print(a%b) #Modulus (remainder after division)
```

```
8
-2
15
0.6
0
243
3
```

## Comparison Operator:

These operators compare two values and return a Boolean value (True or False):

```
a = 3
b = 5
print(a<b) #Less than
print(a>b) #Greater than
print(a>=b) #Greater than or equal to
print(a<=b) #Less than or equal to
print(a==b) #Equal
print(a!=b) #Not equal
```

```
True
False
False
True
False
True
```

## Assignment Operators:

These operators are used to assign values to variables:

```
a = 3
b = 5
a += b #Add and assign
print(a)

a -= b #Subtract and assign
print(a)

a *= b #Multiply and assign
print(a)

a /= b #Divide and assign
print(a)

a %= b #Modulus and assign
print(a)

a **= b #Exponentiate and assign
print(a)

a //= b #Floor divide and assign
print(a)
```

```
8
3
15
3.0
3.0
243.0
48.0
```

## Logical Operators:

These operators are used to combine conditional statements:

```
a, b = True, False

print(a and b) #Returns True if both statements are true

print(a or b) #Returns True if one of the statements is true

print(not b) #Reverses the result, returns True if the result is false
```

```
False
True
True
```

## Bitwise Operator:

These operators perform bit-level operations on binary numbers:

```
a, b = 34, 45

print(a & b) #AND

print(a | b) #OR

print(a ^ b) #XOR

print(~ b) #NOT

print(a >> b) #Right Shift

print(a << b) #Left Shift
```

```
32
47
15
-46
0
1196268651020288
```

## Special Operators

### Identity Operator:

These operators compare the memory locations of two objects:

- `is`: Returns True if both variables are the same object
- `is not`: Returns True if both variables are not the same object

```
a = 5
b = 5
print(a is b)

l1 = [1,2,3]
l2 = [1,2,3]
print(l1 is l2)

e1 = "DDIC"
e2 = "DDIC"
print(e1 is not e2)
```

```
True
False
False
```

### Membership Operator:

These operators test if a sequence contains a certain value:

- `in`: Returns True if a sequence with the specified value is present in the object
- `not in`: Returns True if a sequence with the specified value is not present in the object

```
abc = [1,2,3,4]
print(2 in abc)
```

```
True
```



## Day 6: Python Flow Control

### Python if...else Statement:

In computer programming, the `if` statement is a conditional statement. It is used to execute a block of code only when a specific condition is met.

#### Syntax:

##### If condition:

**# body of if statement**

```
number = int(input('Enter a number: '))

if number > 0:
    print(f'{number} is a positive number.')

print('A statement outside the if statement.')
```

```
Enter a number: 34
34 is a positive number.
A statement outside the if statement.
```

```
number = int(input('Enter a number: '))

if number > 0:
    print(f'{number} is a positive number.')

print('A statement outside the if statement.')
```

```
Enter a number: -4
A statement outside the if statement.
```

## Python If\_else statement:

An **if** statement can have an optional **else** clause. The **else** statement executes if the condition in the **if** statement evaluates to **False**.

### Syntax:

#### If condition:

**# body of if statement**

#### else:

**# body of else statement**

```
number = int(input('Enter a number: '))

if number > 0:
    print('Positive number')
else:
    print('Not a positive number')
```

Enter a number: 5  
Positive number

```
number = int(input('Enter a number: '))

if number > 0:
    print('Positive number')
else:
    print('Not a positive number')
```

Enter a number: -4  
Not a positive number

## Python if\_elif\_else statement:

The if...else statement is used to execute a block of code among two alternatives. However, if we need to make a choice between more than two alternatives, we use the if...elif...else statement.

```
number = -5

if number > 0:
    print('Positive number')

elif number < 0:
    print('Negative number')

else:
    print('Zero')
```

Negative number

## Python nested if statement:

```
number = 5

# outer if statement
if number >= 0:
    # inner if statement
    if number == 0:
        print('Number is 0')

    # inner else statement
    else:
        print('Number is positive')

# outer else statement
else:
    print('Number is negative')
```

Number is positive

## For Loop:

The for loop is used to iterate over a sequence (like a list, tuple, string, or range).

```
# Example of for loop  
numbers = [1, 2, 3, 4, 5]  
for num in numbers:  
    print(num)
```

```
1  
2  
3  
4  
5
```

```
# Example of for loop with range  
for i in range(5):  
    print(i)
```

```
0  
1  
2  
3  
4
```

## While Loop:

The while loop is used to execute code as long as a condition is true.

```
# Example of while loop  
count = 0  
while count < 5:  
    print(count)  
    count += 1
```

```
0  
1  
2  
3  
4
```

```
# Example of while loop with break statement  
count = 0  
while True:  
    print(count)  
    count += 1  
    if count >= 5:  
        break
```

```
0  
1  
2  
3  
4
```

## Break and continue Statement:

The break statement is used to exit a loop, while the continue statement skips the rest of the code inside a loop for the current iteration and continues with the next iteration.

```
# Example of break statement  
for i in range(10):  
    if i == 5:  
        break  
    print(i)
```

```
0  
1  
2  
3  
4
```

```
# Example of continue statement  
for i in range(10):  
    if i == 5:  
        continue  
    print(i)
```

```
0  
1  
2  
3  
4  
6  
7  
8  
9
```

## References:

**Python downloading site:** [Download Python | Python.org](#)

**Anaconda site:** [Download Now | Anaconda](#)

**YouTube link to resolve getting error while opening Jupiter**

**Notebook:** <https://www.youtube.com/watch?v=4V2FSeARCSQ>

**Python Language course:** [Scaler Topics](#)

**Programiz website for example:** [Learn Python Programming](#)