Data Science with Python

By Vaishali KUNJIR

About Me

Vaishali KUNJIR

Lead Data Engineer | Data Science R & D Engineer | Senior software Engineer | Full stack developer(Angular, Python) | Automation | Django | Flask | Frontend | Python expert | Scrum Master| project manager | Fluent French speaker

Education:

- Master in Machine learning specialization in automatic image and natural language processing

  Université de Caen Normandie, Caen, France
- Master of Computer Science

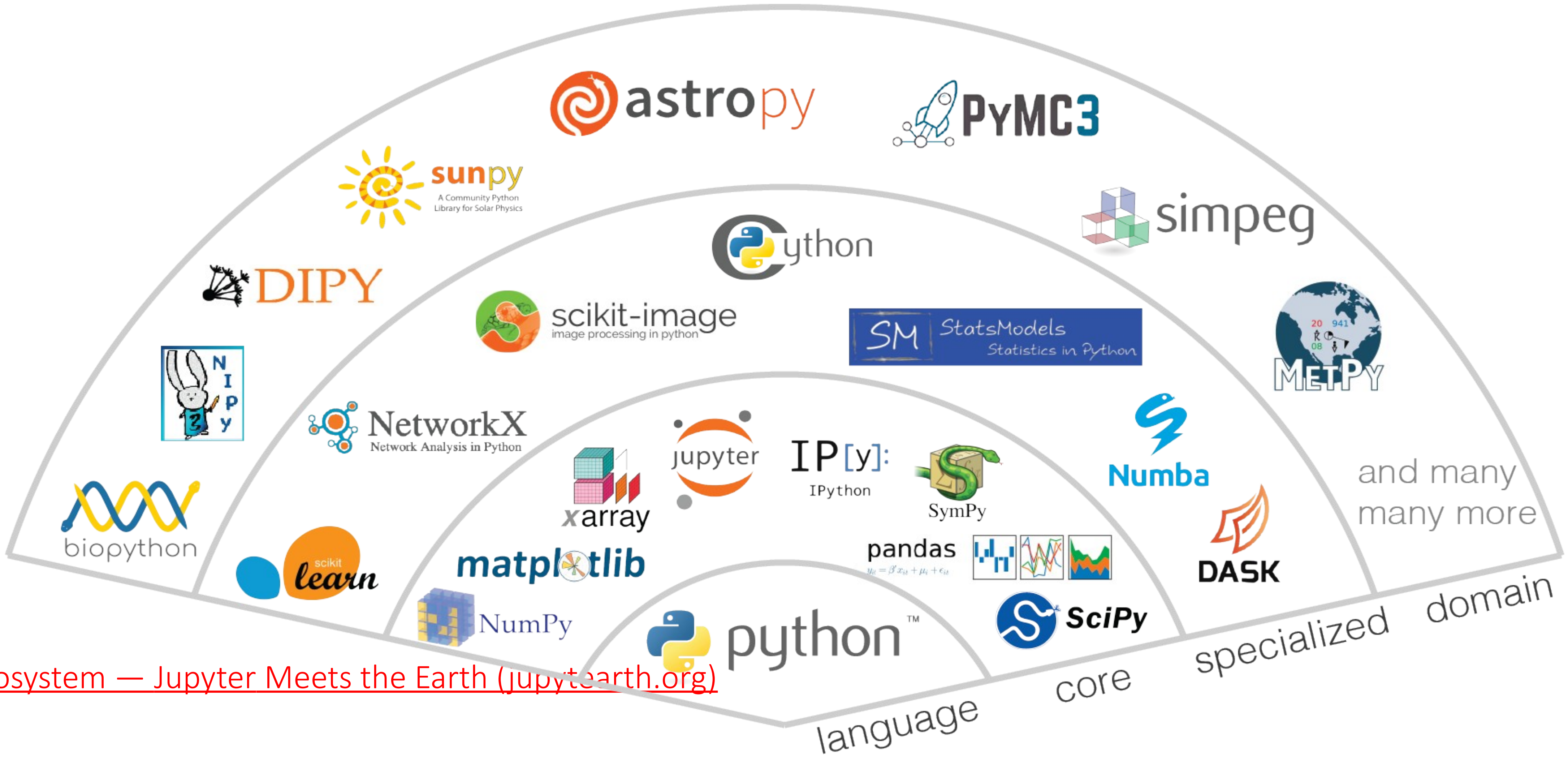
  Savitribai Phule Pune University, Pune, India

CERTIFICATION :

- Agile with Atlassian Jira MOOC Coursera
- Atlassian Community Scrum master certification
- Machine learning MOOC Coursera, Stanford University
- Advanced Diploma in software engineering

  CAT, Pune, India FLE A2 French language

Python Ecosystem



** Reference: Ecosystem — Jupyter Meets the Earth (jupytearth.org)

Introduction to Python

- What is python?
- Scope of Python language (Job opportunities and companies)
- Setup of Development environment
- Basic Syntax, Comments, Variables, Data Types, Operators, Decision Making, Loops, Numbers, Data Structures (Numbers, String, Lists, Tuples, Dictionary, Date & Time), Functions, Modules, File I/O, Exceptions
- More information about python, Please refer ➔ The Python Tutorial — Python 3.11.4 documentation

What is python?

Python is a very popular general-purpose interpreted, interactive, object-oriented, and high-level programming language. Python is dynamically-typed and garbage-collected programming language. It was created by Guido van Rossum during 1985- 1990. Like Perl, Python source code is also available under the GNU General Public License (GPL)

Why to learn Python?

- Python is Open Source which means its available free of cost.
- Python is simple and so easy to learn
- Python is versatile and can be used to create different web applications, AI/ML algorithms, Data analysis, Scripts etc.
- Python has powerful development libraries include AI, ML etc.
- Python is much in demand and ensures high salary

Characteristics of Python

Following are important characteristics of **Python Programming** −

- It supports functional and structured programming methods as well as OOP.
- It can be used as a scripting language or can be compiled to byte-code for building large applications.
- It provides very high-level dynamic data types and supports dynamic type checking.
- It supports automatic garbage collection.
- It can be easily integrated with C, C++, COM, ActiveX, CORBA, and Java.
- Python has been ported to the Java and .NET virtual machines

Scope of Python language (Job opportunities and Companies)

➢ Job Opportunities

- Game developer
- Web designer
- Python developer
- Full-stack developer
- Machine learning engineer
- Data scientist
- Data analyst
- Data engineer
- DevOps engineer
- Software engineer
- Many more other roles

➢ Companies

- Google
- Intel
- NASA
- PayPal
- Facebook
- IBM
- Amazon
- Netflix
- Pinterest
- Uber
- Many more...

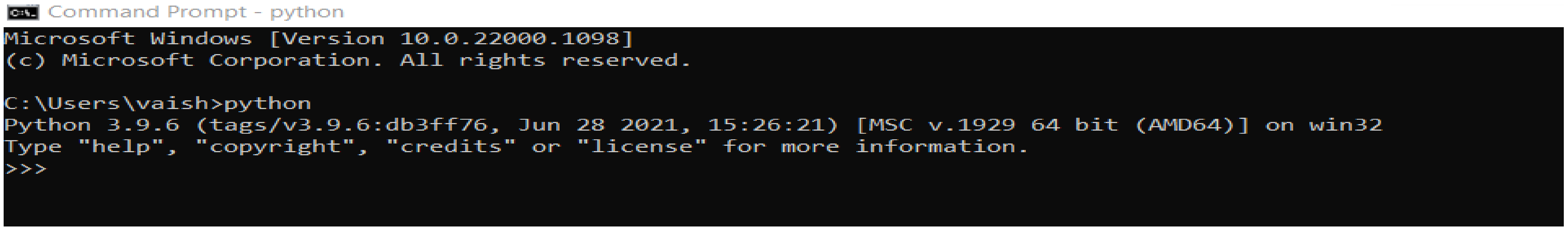
Setup of Development environment

Python is available on a wide variety of platforms. Following is the list of platforms which are compatible with python:

- Unix (Solaris, Linux, FreeBSD, AIX, HP/UX, SunOS, IRIX, etc.)
- Linux
- Windows
- Mac OS X
- Win 9x/NT/2000
- Macintosh (Intel, PPC, 68K)
- OS/2
- DOS (multiple versions)
- PalmOS
- Nokia mobile phones
- Windows CE
- Acorn/RISC OS
- BeOS
- Amiga
- VMS/OpenVMS
- QNX
- VxWorks
- Psion

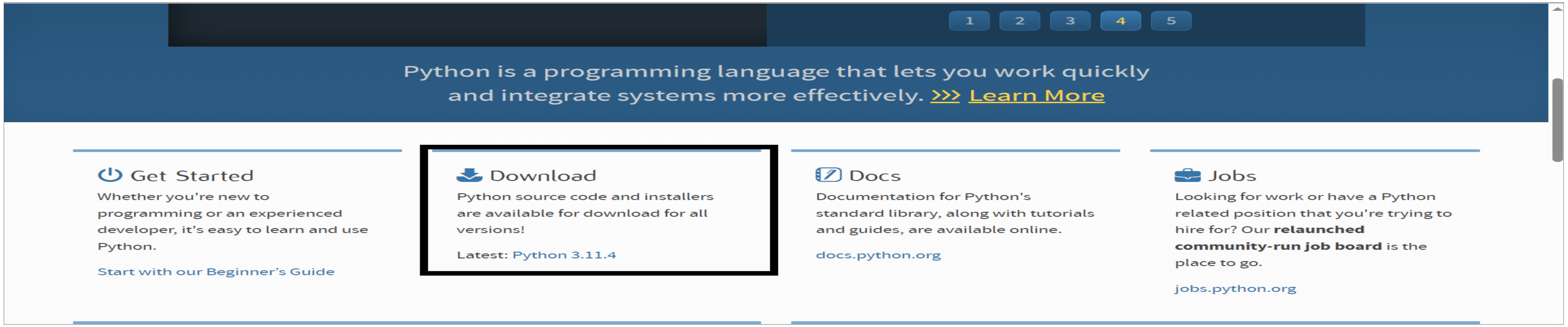
Setup of Development environment

- Open command prompt, type **python.** If you see following output, then it means that, python is installed.



```
Command Prompt - python
Microsoft Windows [Version 10.0.22000.1098]
(c) Microsoft Corporation. All rights reserved.

C:\Users\vaish>python
Python 3.9.6 (tags/v3.9.6:db3ff76, Jun 28 2021, 15:26:21) [MSC v.1929 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

- If no output is displayed, means python installation is required
- Go to www.python.org ➔ Downloads



Python is a programming language that lets you work quickly and integrate systems more effectively. >>> Learn More

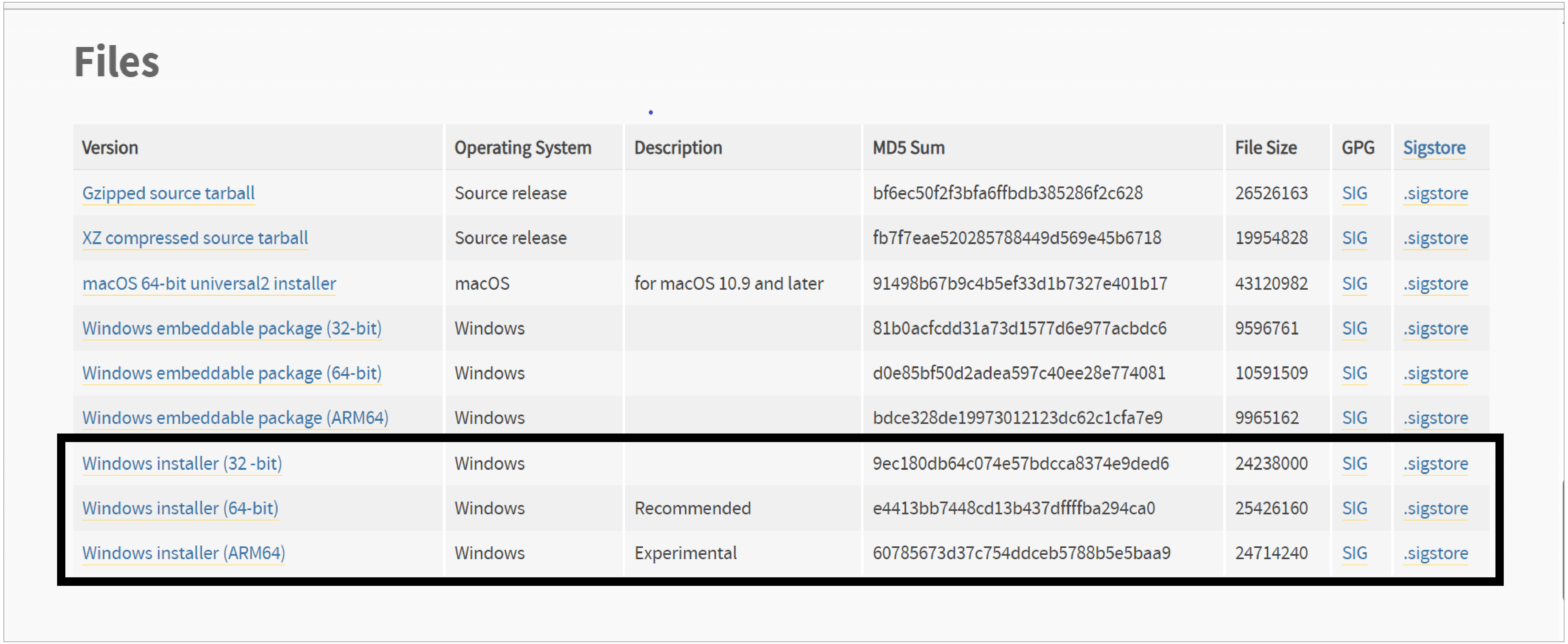
Get Started
Whether you're new to programming or an experienced developer, it's easy to learn and use Python.
Start with our Beginner's Guide

Download
Python source code and installers are available for download for all versions!
Latest: Python 3.11.4

Docs
Documentation for Python's standard library, along with tutorials and guides, are available online.
docs.python.org

Jobs
Looking for work or have a Python related position that you're trying to hire for? Our **relaunched community-run job board** is the place to go.
jobs.python.org

Setup of Development environment

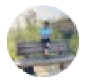


Files

Version	Operating System	Description	MD5 Sum	File Size	GPG	Sigstore
Gzipped source tarball	Source release		bf6ec50f2f3bfa6ffbdb385286f2c628	26526163	SIG	.sigstore
XZ compressed source tarball	Source release		fb7f7eae520285788449d569e45b6718	19954828	SIG	.sigstore
macOS 64-bit universal2 installer	macOS	for macOS 10.9 and later	91498b67b9c4b5ef33d1b7327e401b17	43120982	SIG	.sigstore
Windows embeddable package (32-bit)	Windows		81b0acfcdd31a73d1577d6e977acbdc6	9596761	SIG	.sigstore
Windows embeddable package (64-bit)	Windows		d0e85bf50d2adea597c40ee28e774081	10591509	SIG	.sigstore
Windows embeddable package (ARM64)	Windows		bdce328de19973012123dc62c1cfa7e9	9965162	SIG	.sigstore
Windows installer (32 -bit)	Windows		9ec180db64c074e57bdcca8374e9ded6	24238000	SIG	.sigstore
Windows installer (64-bit)	Windows	Recommended	e4413bb7448cd13b437dffffba294ca0	25426160	SIG	.sigstore
Windows installer (ARM64)	Windows	Experimental	60785673d37c754ddceb5788b5e5baa9	24714240	SIG	.sigstore

Python Installation with Anaconda/Jupyter Notebook

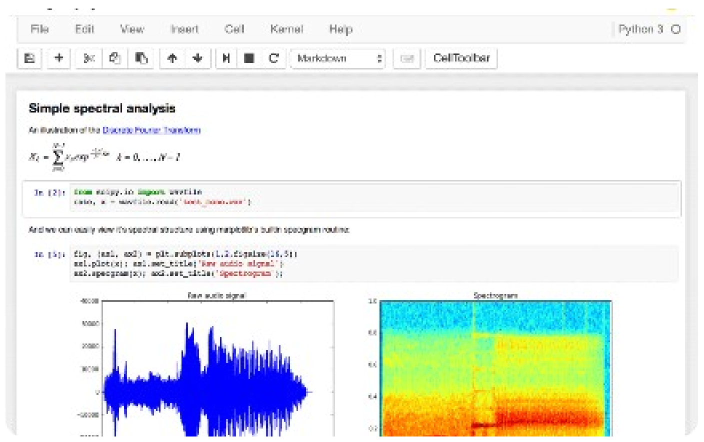
Learn the benefits of using Anaconda and jupyter Notebook for Python installation. Simplified process, easy package management, and integrated development environment.



by vaishali kunjir

Introduction to Anaconda

Get acquainted with Anaconda, the Python distribution that provides a comprehensive package manager and environment management system.



Simple spectral analysis

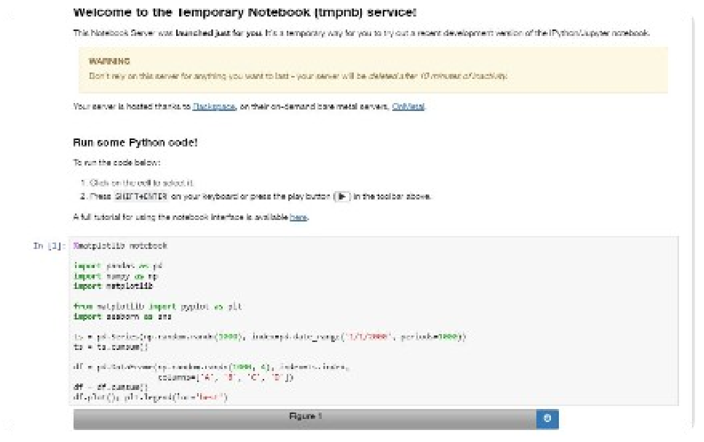
An illustration of the Discrete Fourier Transform

$$X_k = \sum_{n=0}^{N-1} x_n \exp^{-\frac{2\pi i}{N} kn} \quad k = 0, \ldots, N-1$$

```
In [2]: from scipy.io import wavfile
        rate, x = wavfile.read('test_mono.wav')
```

And we can easily view it's spectral structure using matplotlib's builtin specgram routine:

```
In [5]: fig, (ax1, ax2) = plt.subplots(1,2,figsize=(16,5))
        ax1.plot(x); ax1.set_title('Raw audio signal')
        ax2.specgram(x); ax2.set_title('Spectrogram');
```



Welcome to the Temporary Notebook (tmpnb) service!

This Notebook Server was **launched just for you**. It's a temporary way for you to try out a recent development version of the IPython/Jupyter notebook.

WARNING
Don't rely on this server for anything you want to last - your server will be *deleted after 10 minutes of inactivity*.

Your server is hosted thanks to Rackspace, on their on-demand bare metal servers, OnMetal.

Run some Python code!

To run the code below:

1. Click on the cell to select it.
2. Press SHIFT+ENTER on your keyboard or press the play button (▶) in the toolbar above.

A full tutorial for using the notebook interface is available here.

```
In [1]: %matplotlib notebook

        import pandas as pd
        import numpy as np
        import matplotlib

        from matplotlib import pyplot as plt
        import seaborn as sns

        ts = pd.Series(np.random.randn(1000), index=pd.date_range('1/1/2000', periods=1000))
        ts = ts.cumsum()

        df = pd.DataFrame(np.random.randn(1000, 4), index=ts.index,
                          columns=['A', 'B', 'C', 'D'])
        df = df.cumsum()
        df.plot(); plt.legend(loc='best')
```

What is Anaconda?

Learn about Anaconda's features, including pre-installed packages and tools for data science.

What is Jupyter Notebook?

Discover the interactive coding interface that jupyter Notebook offers, enabling documentation and computational elements.

Jupyter Notebook Applications

Explore the various applications of jupyter Notebook, such as data exploration, visualization, and machine learning.

Step-by-Step Installation Guide

Downloading Anaconda

Access the official Anaconda website and obtain the installation file for your operating system.

Installing Anaconda

Follow the installation wizard instructions to set up Anaconda on your computer.

Setting up the Environment

Configure the Anaconda environment variables to ensure seamless integration with your system.

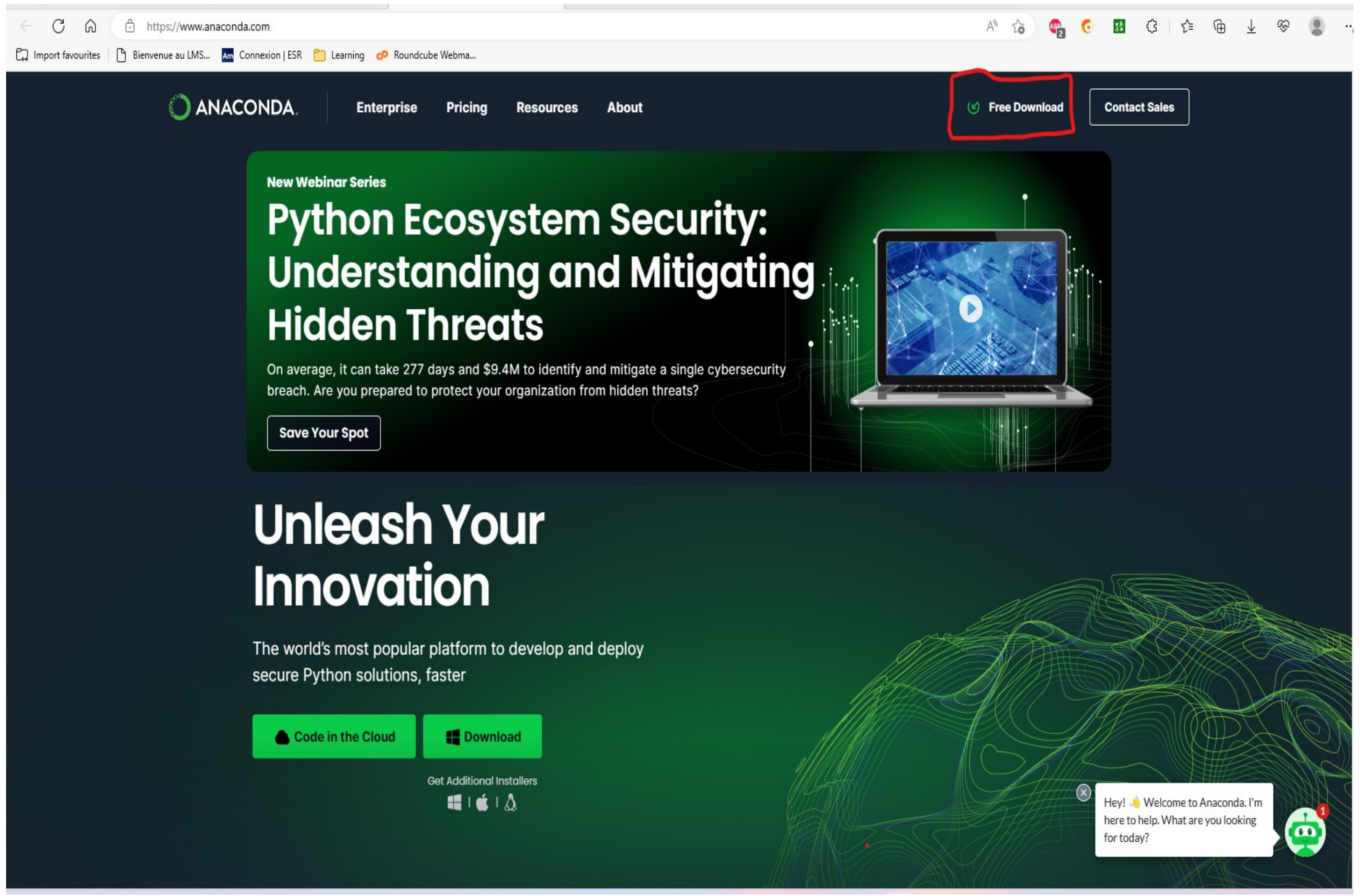
Verifying the Installation

Confirm that Anaconda has been installed correctly by running a simple Python command in the terminal.

Anaconda installation

Browse website https://www.anaconda.com/ and click on **Free Download. Click on Download button to download Anaconda distribution.**

Refer this installation guide https://docs.anaconda.com/free/anaconda/install/index.html



New Webinar Series

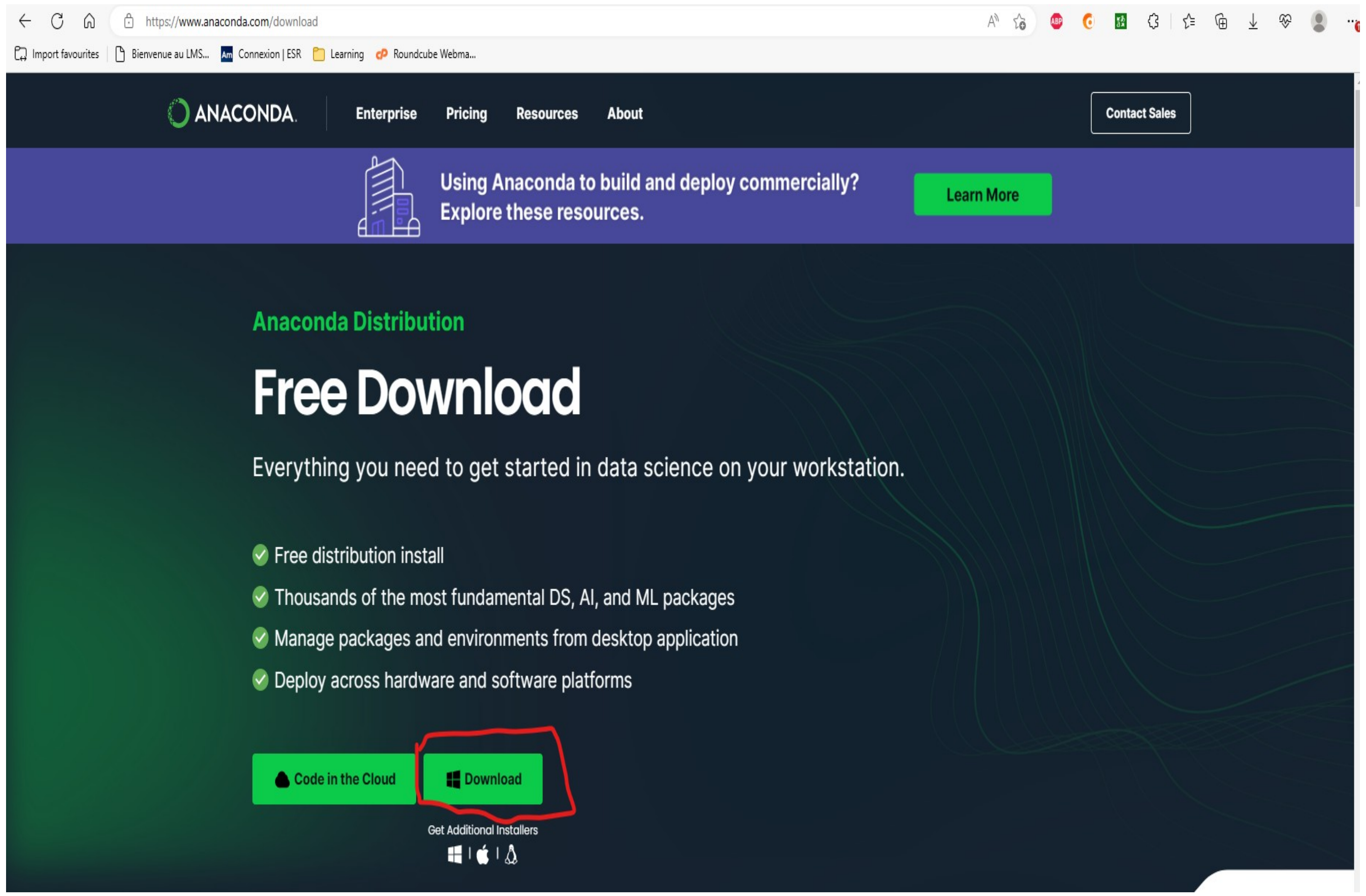
Python Ecosystem Security: Understanding and Mitigating Hidden Threats

On average, it can take 277 days and $9.4M to identify and mitigate a single cybersecurity breach. Are you prepared to protect your organization from hidden threats?

Unleash Your Innovation

The world's most popular platform to develop and deploy secure Python solutions, faster

Hey! 👋 Welcome to Anaconda. I'm here to help. What are you looking for today?



Using Anaconda to build and deploy commercially? Explore these resources.

Anaconda Distribution

Free Download

Everything you need to get started in data science on your workstation.

- Free distribution install
- Thousands of the most fundamental DS, AI, and ML packages
- Manage packages and environments from desktop application
- Deploy across hardware and software platforms

Exploring Jupyter Notebook

1 **Opening Jupyter Notebook**

Learn how to launch Jupyter Notebook from the command line or Anaconda Navigator.

2 **Creating and Running Scripts**

Discover how to create and execute Python scripts within the Jupyter Notebook environment.

3 **Saving and Sharing Work**

Understand how to save your Jupyter Notebook projects and share them with others as interactive documents.

YY

Conclusion and Next Steps

1 **Recap of the Installation Process**

Download and install Anaconda

2 **Further Resources**

https://learning.anaconda.cloud/jupyter-notebook-basics

Variables, Data Types, Operators and Operands

Welcome to the world of Python! In this presentation, we'll explore the magic of variables, data types, operators and operands in Python and learn how to create beautiful programs as if they were a piece of art.

What are Variables in Python?

1 **Definition**

Variables are like containers that hold data in a computer program.

2 **Importance**

They enable us to store and manipulate data in a program, making it more flexible and powerful.

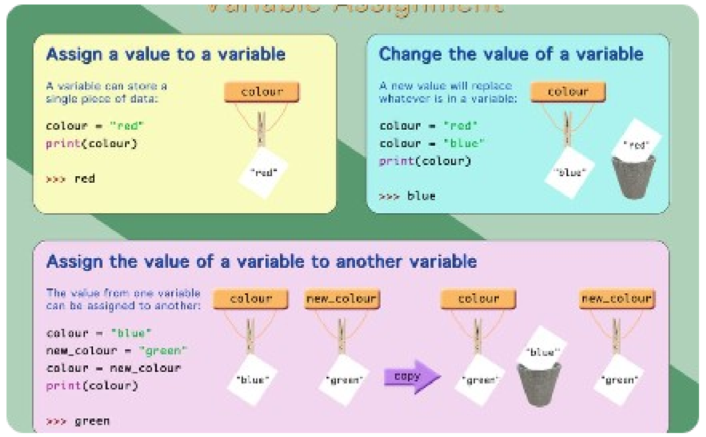
3 **Declaration**

We declare variables by assigning a name to them using the syntax "variable_name = value".

4 **Assignment**

We assign values to variables using the equals (=) sign.

Manipulating Variables in Python


Assign a value to a variable
A variable can store a single piece of data:
colour
"red"
Change the value of a variable
A new value will replace whatever is in a variable:
colour
"blue"
"red"
Assign the value of a variable to another variable
The value from one variable can be assigned to another:
colour
new_colour
"blue"
"green"
copy
colour
new_colour
"green"
"blue"
"green"

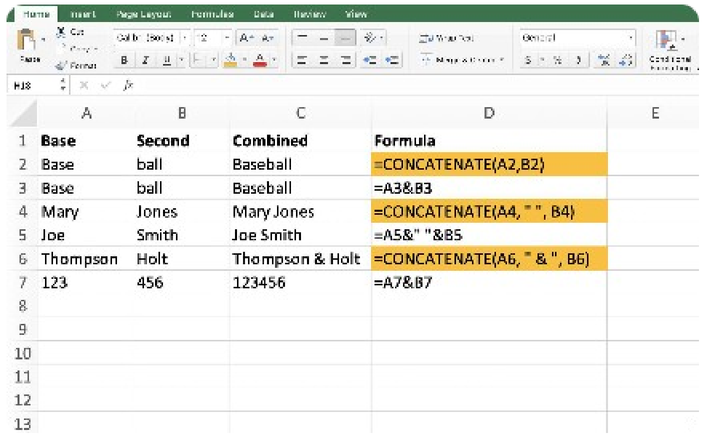
```
colour = "red"
print(colour)

>>> red
```

```
colour = "red"
colour = "blue"
print(colour)

>>> blue
```

```
colour = "blue"
new_colour = "green"
colour = new_colour
print(colour)

>>> green
```

Basic Operations

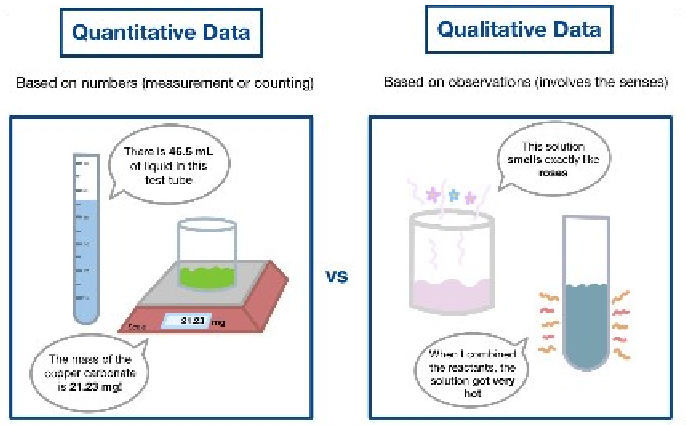
Variables can be added, subtracted, multiplied, and divided just like numbers.



	A	B	C	D	E
1	Base	Second	Combined	Formula	
2	Base	ball	Baseball	=CONCATENATE(A2,B2)	
3	Base	ball	Baseball	=A3&B3	
4	Mary	Jones	Mary Jones	=CONCATENATE(A4, " ", B4)	
5	Joe	Smith	Joe Smith	=A5&" "&B5	
6	Thompson	Holt	Thompson & Holt	=CONCATENATE(A6, " & ", B6)	
7	123	456	123456	=A7&B7	
8					
9					
10					
11					
12					
13					

Concatenation

Variables that store strings can be combined using the "+" operator.


Quantitative Data
Based on numbers (measurement or counting)
There is 46.5 mL of liquid in this test tube
21.23 mg
The mass of the copper carbonate is 21.23 mg!
vs
Qualitative Data
Based on observations (involves the senses)
This solution smells exactly like roses
When I combined the reactants, the solution got very hot

Comparisons

Variables can be compared using logical operators like "==", "!=", "<", ">" and "<=, ">=".

Scope and Lifetime of Variables

Scope

Variables have a scope, which determines where they can be accessed in a program.

For example, a variable declared inside a function can only be accessed from within the function.

Lifetime

The lifetime of a variable is the period during which it exists in memory while the program is running. Once a variable goes out of scope, it may be deleted by the interpreter to free up memory.

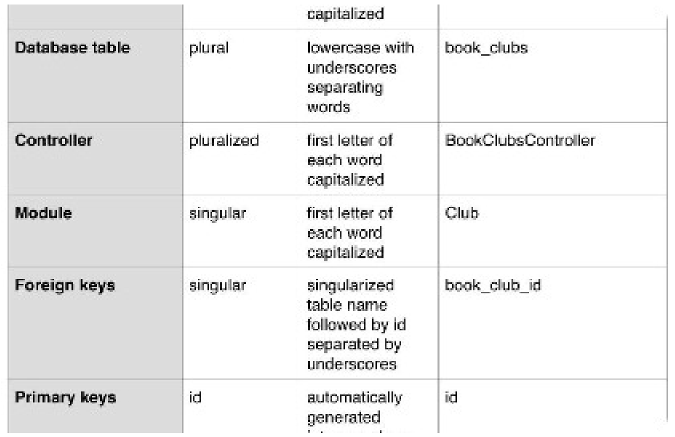
Global

Global variables are accessible from anywhere in the program and can be useful for storing values that need to be accessed in multiple places.

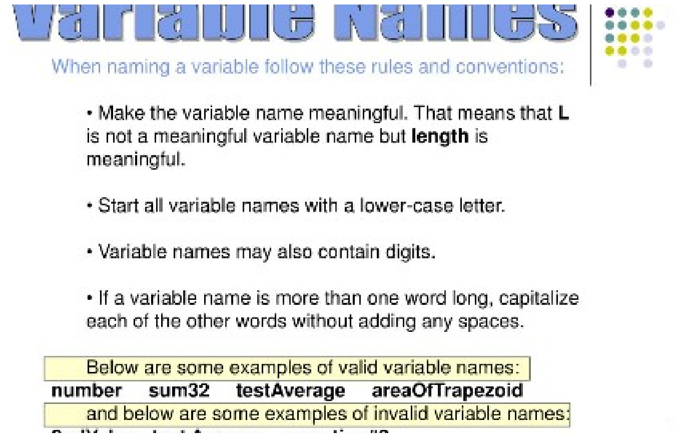
Local

Local variables are only accessible from within the block of code in which they are declared.

Best Practices for Naming Variables



		capitalized	
Database table	plural	lowercase with underscores separating words	book_clubs
Controller	pluralized	first letter of each word capitalized	BookClubsController
Module	singular	first letter of each word capitalized	Club
Foreign keys	singular	singularized table name followed by id separated by underscores	book_club_id
Primary keys	id	automatically generated	id



Variable Names

When naming a variable follow these rules and conventions:

• Make the variable name meaningful. That means that **L** is not a meaningful variable name but **length** is meaningful.

• Start all variable names with a lower-case letter.

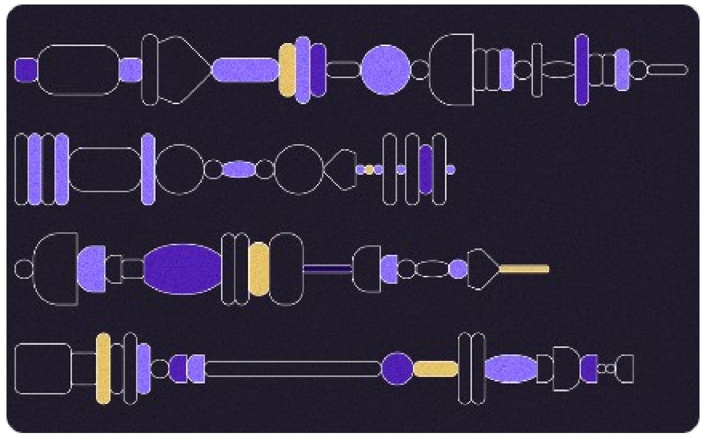
• Variable names may also contain digits.

• If a variable name is more than one word long, capitalize each of the other words without adding any spaces.

Below are some examples of valid variable names:

number sum32 testAverage areaOfTrapezoid

and below are some examples of invalid variable names:



Naming convention

Variables should have meaningful names that reflect their purpose. Use lowercase letters for variable names and use underscores to separate words.

Meaningful names

Choose names that are descriptive and easy to understand. Avoid abbreviations and overly generic names.

Length

Keep variable names short and sweet. Long names can be difficult to read and prone to typos.

Python Reserved Words

The following list shows the Python keywords. These are reserved word’s and you cannot use them as constant or variable or any other identifier names. All the Python keywords contain lowercase letters only.

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

Data Types: The Two Sides of the Binary World

Python data types include numbers, strings, booleans, lists, tuples, and dictionaries. Each data type has unique functions and properties, and learning to use them effectively is crucial for intricate programming.

Strings

Strings are sequences of Unicode characters. They are mutable and can be tweaked, manipulated, and concatenated to create new strings or alter existing ones.

Booleans

Boolean data types represent two values: True or False. They are often used to test conditions and validate if statements.

Dicts, Lists, Tuples, Set

Python's dictionaries and lists are mutable and can store a collection of any type of information we want. They are incredibly flexible and useful in more complex programs. Tuples are immutable

Data Types Declaration

1) Integers (int): Integers are whole numbers, positive or negative, without any decimal point.

Example:
age = 30

2) Floating-Point Numbers (float): Floating-point numbers, or floats, represent real numbers and can have decimal points.

Example:
Pi = 3.14

3) Strings (str): Strings are sequences of characters, and they are used to represent text in Python. Strings are enclosed in either single (') or double (") quotes.

Example:
name = "Alice"

4) Boolean (bool): Boolean data type represents two values: True and False. Booleans are often used in conditional and loop statements.

Example:
is_adult = True

Data Types Declaration

5) Lists (list): Lists are ordered collections of items, which can be of different data types. Lists are defined by square brackets [ ]

Example:

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

6) Tuples (tuple): Tuples are similar to lists, but they are immutable, meaning their elements cannot be changed after creation. Tuples are defined by parentheses ( )

Example:

Coordinates = (3, 4)

7) Dictionaries (dict): Dictionaries are collections of key-value pairs. They are unordered and can be used to store data values like a map. Dictionaries are defined by curly braces { }

Example:

person = {"name": "Bob", "age": 25, "city": "New York"}

8) Sets: You can create a set by placing a comma-separated sequence of items inside curly braces {}, or by using the set() constructor.

Example:

# Creating a set with curly braces

my_set = {1, 2, 3, 4, 5}

# Creating a set with set() constructor

another_set = set([3, 4, 5, 6, 7])

Operators and Operands: The Rhythm of Your Code

Operators and operands execute functions on data to return a value.

1. **Arithmetic Operators**

   Mathematical functions like addition, subtraction, multiplication, division and modulus.

2. **Comparison Operators**

   Comparison functions like equal to, greater than, less than, etc.

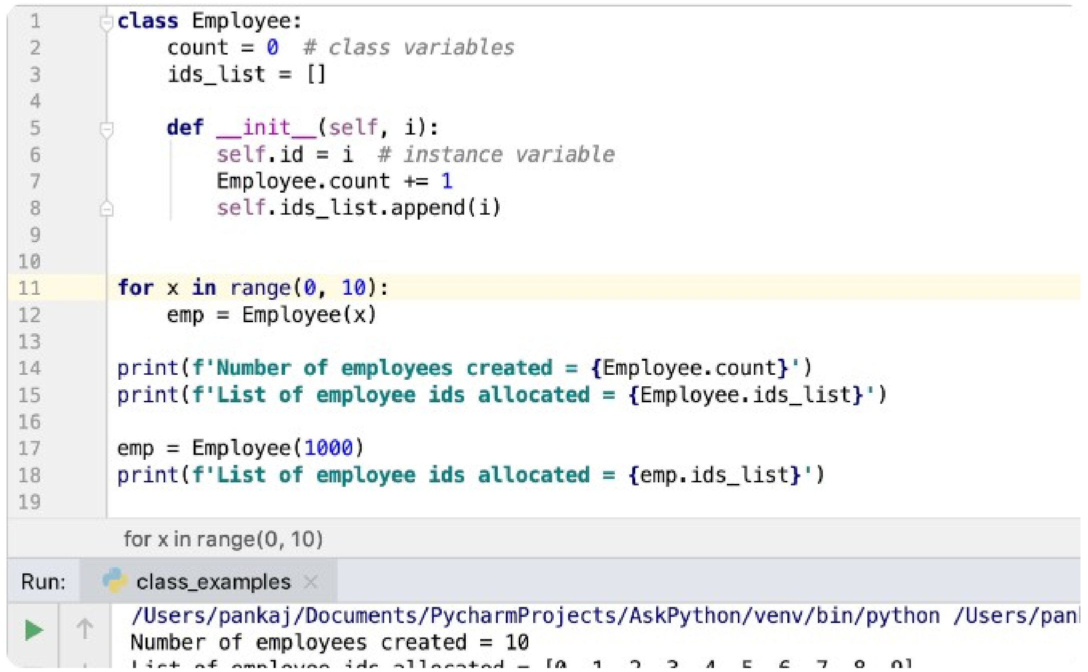
3. **Assignment Operators**

   Operators that assign values, like incrementing and decrementing.

4. **Logical and Bitwise Operators**

   Operators that compare values to a truth table to determine whether a statement is true or false.

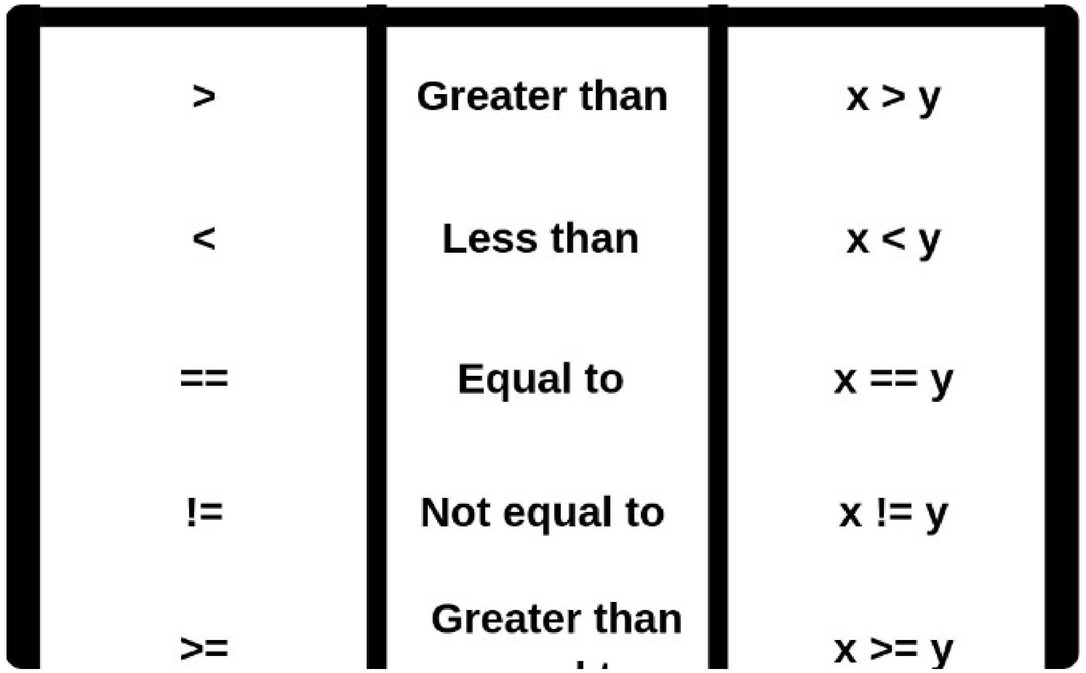
Identity Operators: Differentiating Between Objects in Memory



```
class Employee:
    count = 0  # class variables
    ids_list = []

    def __init__(self, i):
        self.id = i  # instance variable
        Employee.count += 1
        self.ids_list.append(i)


for x in range(0, 10):
    emp = Employee(x)

print(f'Number of employees created = {Employee.count}')
print(f'List of employee ids allocated = {Employee.ids_list}')

emp = Employee(1000)
print(f'List of employee ids allocated = {emp.ids_list}')
```

```
/Users/pankaj/Documents/PycharmProjects/AskPython/venv/bin/python /Users/pan
Number of employees created = 10
```

for x in range(0, 10)
Run:
class_examples



>	Greater than	x > y
<	Less than	x < y
==	Equal to	x == y
!=	Not equal to	x != y
>=	Greater than	x >= y

is vs. ==

The "is" operator checks for object identity, while the "==" operators check for object similarity.

Identity vs. Equality

The "id()" function serves to identify an object in memory, while the "==" operator checks for equality of values.

Membership Operators: Getting Inside the List

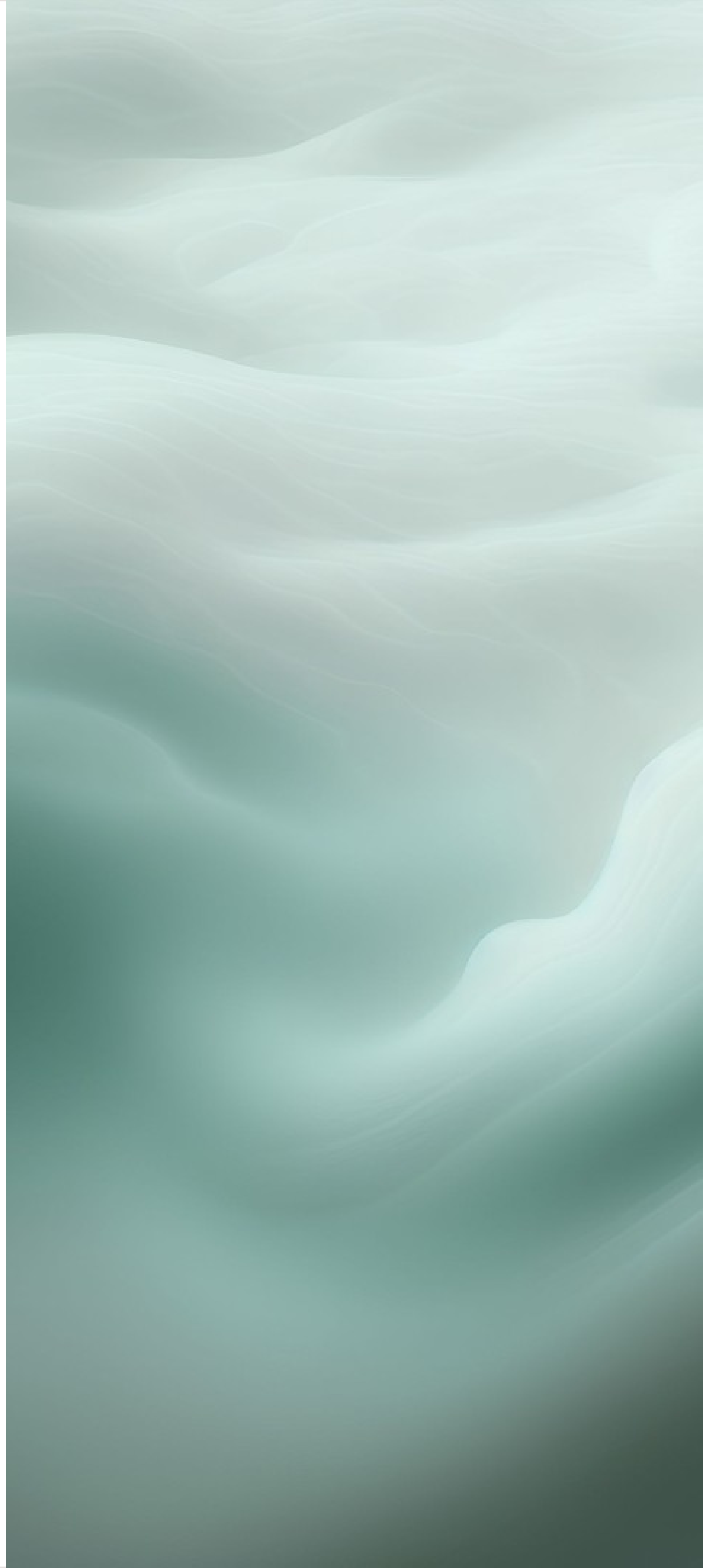
Membership operators check whether a value exists within a sequence, like a list or string.

in

Determines whether the value is present in the object.

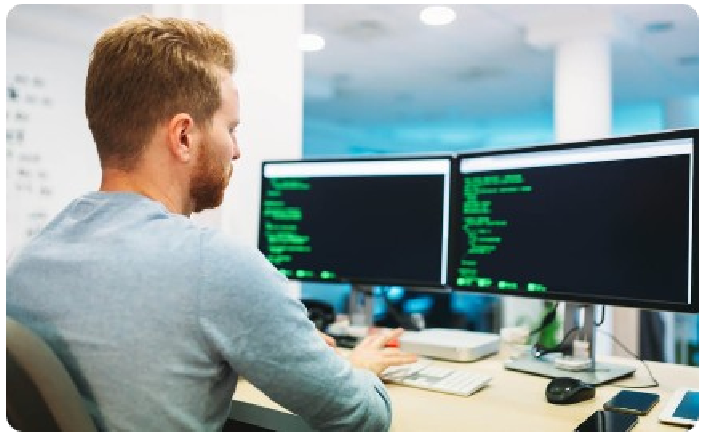
not in

Determines whether the value is not present in the object.

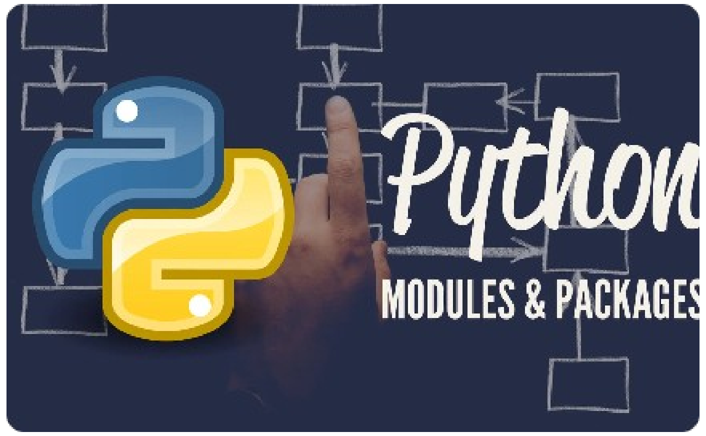


Pro Tips for Pythonic Programs

Make your code more efficient and pythonic with these expert tips!



Color Coded Alphabet
Color the dots in the uppercase letters with the colors used in the matching lowercase letters.
A B C D E F G H I J
K L M N O P Q R S T
U V W X Y Z
a b c d e f g h i j k l m n o p q r s t u v w x y z



Clean Code

Keep your code concise and organized, with comments explaining your reasoning.

Readability

Make sure your code is easy to read, with clear variable names and logical structure.

Useful Libraries

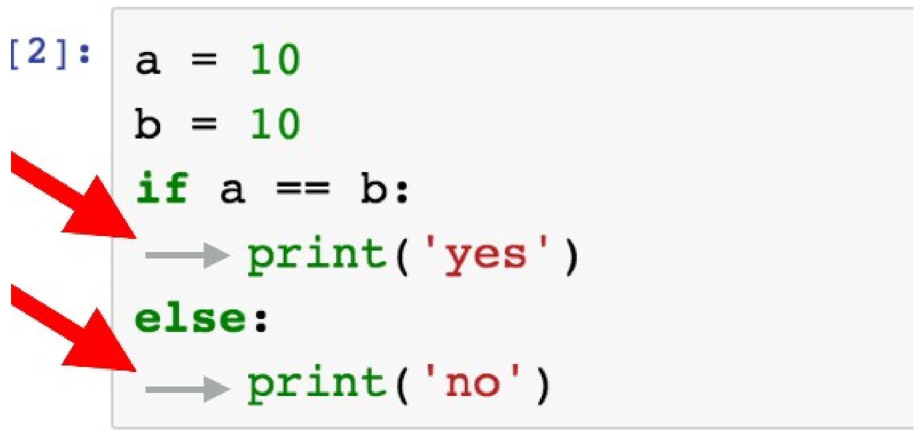
Python has a vast array of libraries such as Numpy, Pandas, Keras designed specifically to solve complex problems. Don't reinvent the wheel!

Understanding Conditional Statements and Loops in Python

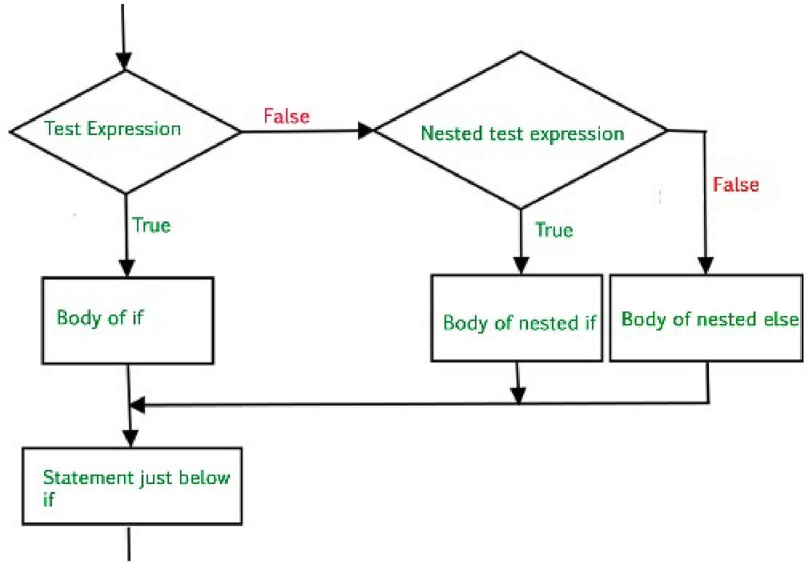
This presentation provides an overview of conditional statements and loops in Python programming. Learn how to use them effectively with the help of examples.

What are Conditional Statements in Python?

In programming, conditional statements are used to make decisions. In Python, it helps to execute certain code only if a condition is satisfied. Explore the syntax and some basic examples of conditional statements in this section.



```
[2]: a = 10
     b = 10
     if a == b:
         print('yes')
     else:
         print('no')
```



If-else Statements

The if-else statement is used to execute different blocks of code depending upon the condition. It is often used in decision making and validation.

Nested If Statements

The nested if statement is used to execute a condition within a condition. It is used when we want to check multiple conditions one after the other.

Decision making / Control Flow

In a real life when certain situation comes and we need to make some decisions and based on these decisions, we decide what should we do next. Similarly, while writing program, we need to make some decision and based on decisions we will execute next block of code.

There are 4 types of control flow statements in python programming:

1. if statement
2. if-else statement
3. nested-if statement
4. if-elif-if ladder

Decision making / Control Flow

if statement

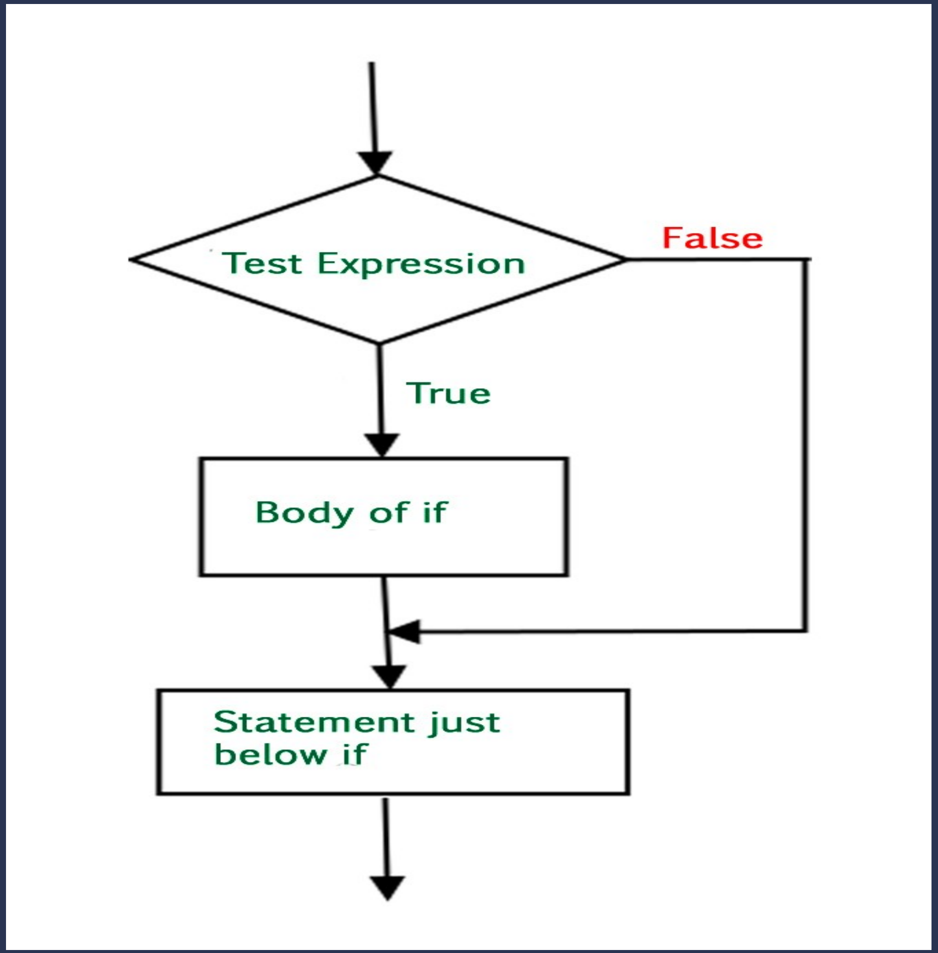
It is used to decide whether a certain statement or block of statements will be executed or not.

Syntax:

```
If  <<condition/Test Expression>>:
       # Executable statements if condition is true
      Statement1
Statement2
```

Example:

```
num = 15
If num>20:
    print(“15 is less than 20”)
print(“Outside if”)
```



Decision making / Control Flow

if-else statement

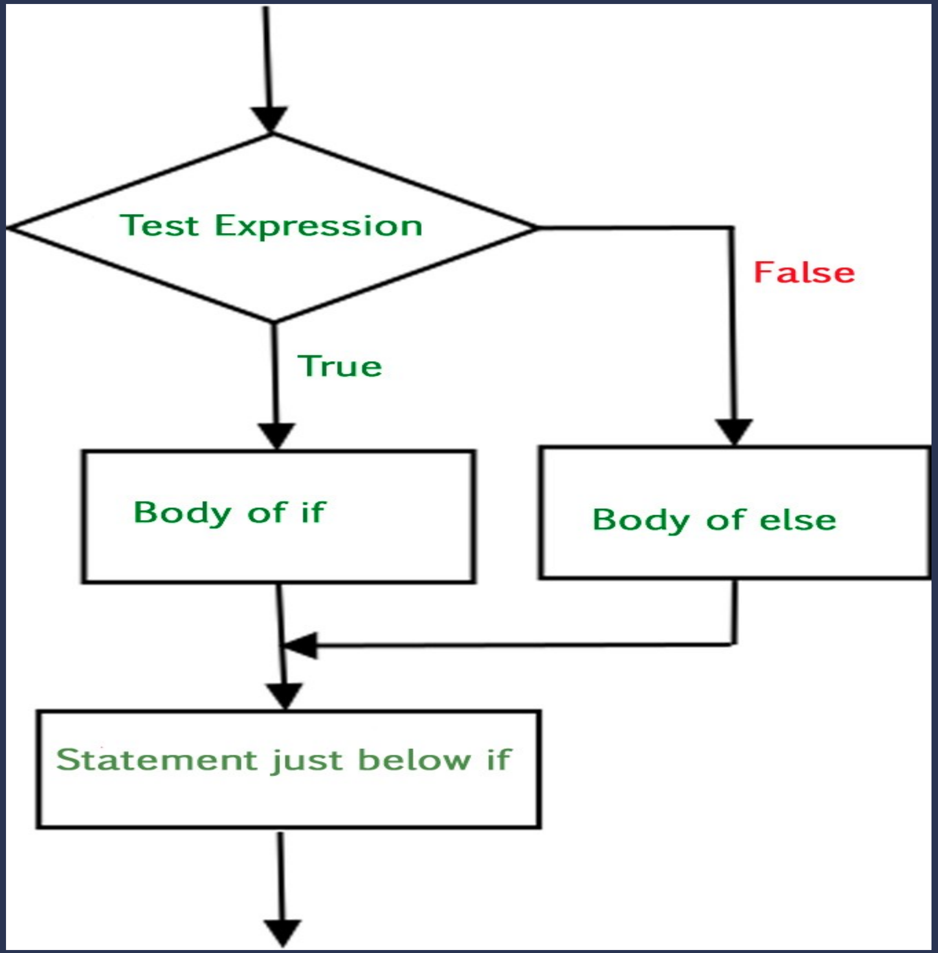
It is used to decide whether a certain statement or block of statements will be executed or not.

Syntax:

```
If <<condition/Test Expression>>:
    # Executable statements if condition is true
    statement1
else:
     #Executes this block if condition is false
     Statement2
```

Example:

```
num = 15
If num<20:
  print("15 is less than 20")
else:
     print("Inside else")
```



Decision making / Control Flow

nested-if statement

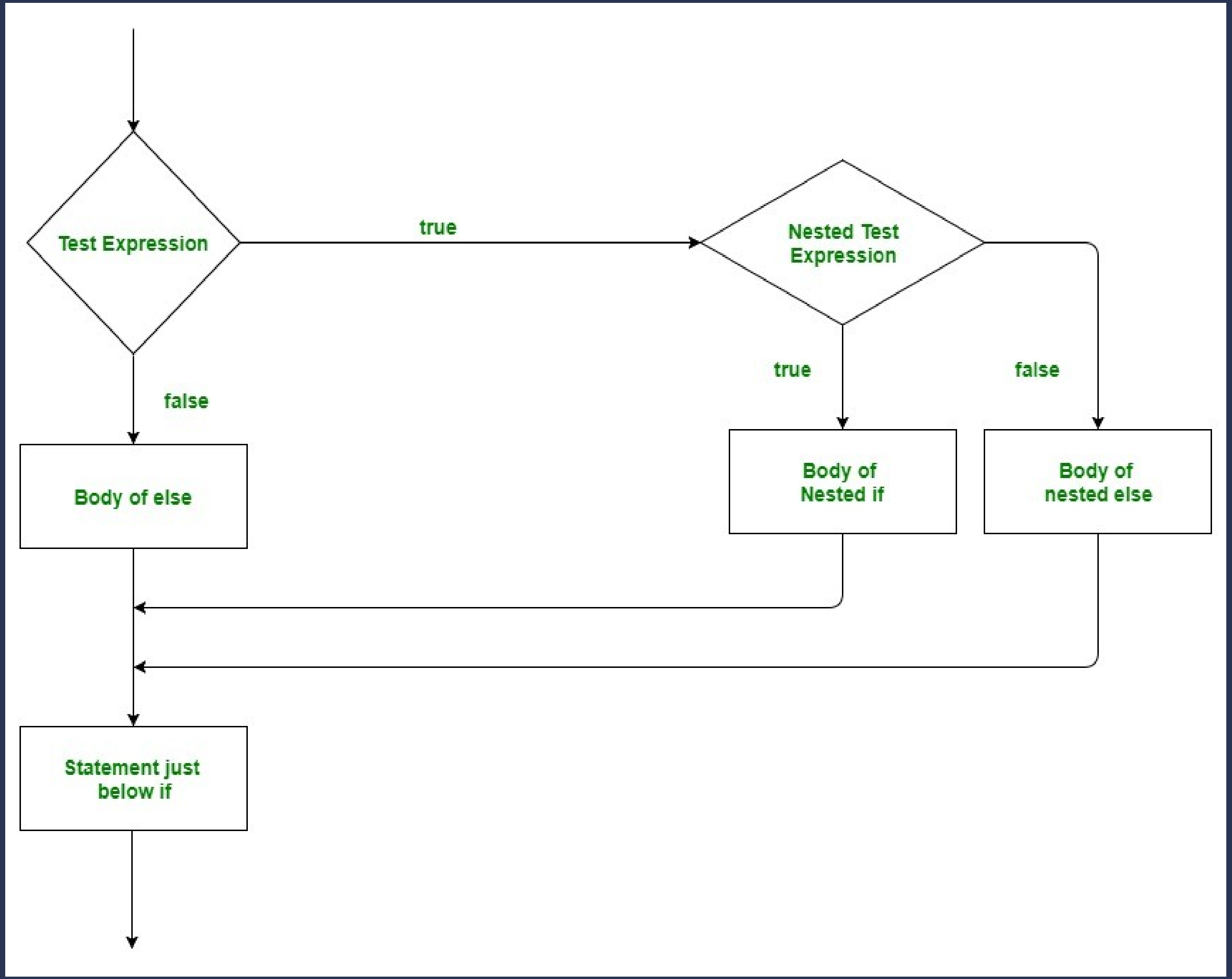
nested if statement means an if statement within an another if statement.

Syntax:

```
If <<condition1/Test Expression1>>:
    # Executable statements if condition1 is true
    Statement1
    If <<Condition2/Test Expression2>>:
        # Executes when condition2 is true
     # if block is end here
# if block end here
```

Example:

```
num = 15
If num == 15:
     # First if statement
     If num < 20:
            print("num is smaller than 20")
```



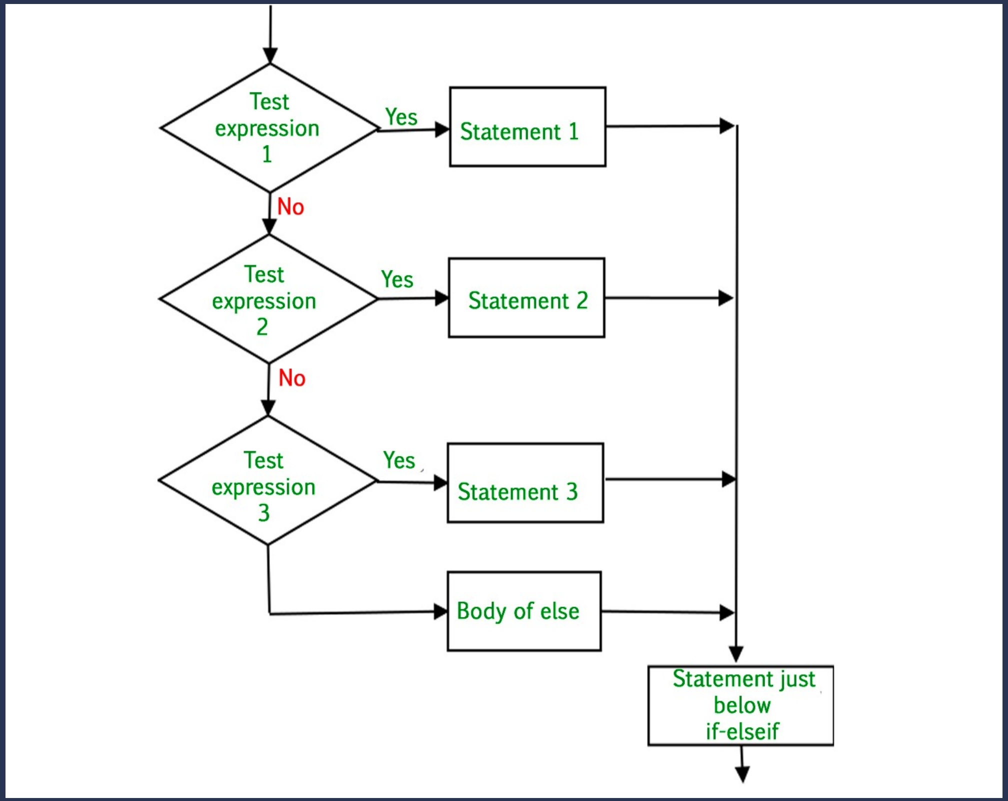
Decision making / Control Flow

if-elif-else ladder

In this ladder, user can decide among multiple options. The if statements are executed from top to bottom. As soon as one of the controlling if Condition is true, the statement associated with that if is executed , and the rest of the ladder is bypassed.

Syntax :

```
If <<condition/Test Expression>>:
    # Executable statements if condition is true
    Statement1
elif <<condition/Test  Expression>>:
    Statement
.
.
else:
     #Executes this block if condition is false
     Statement2
```

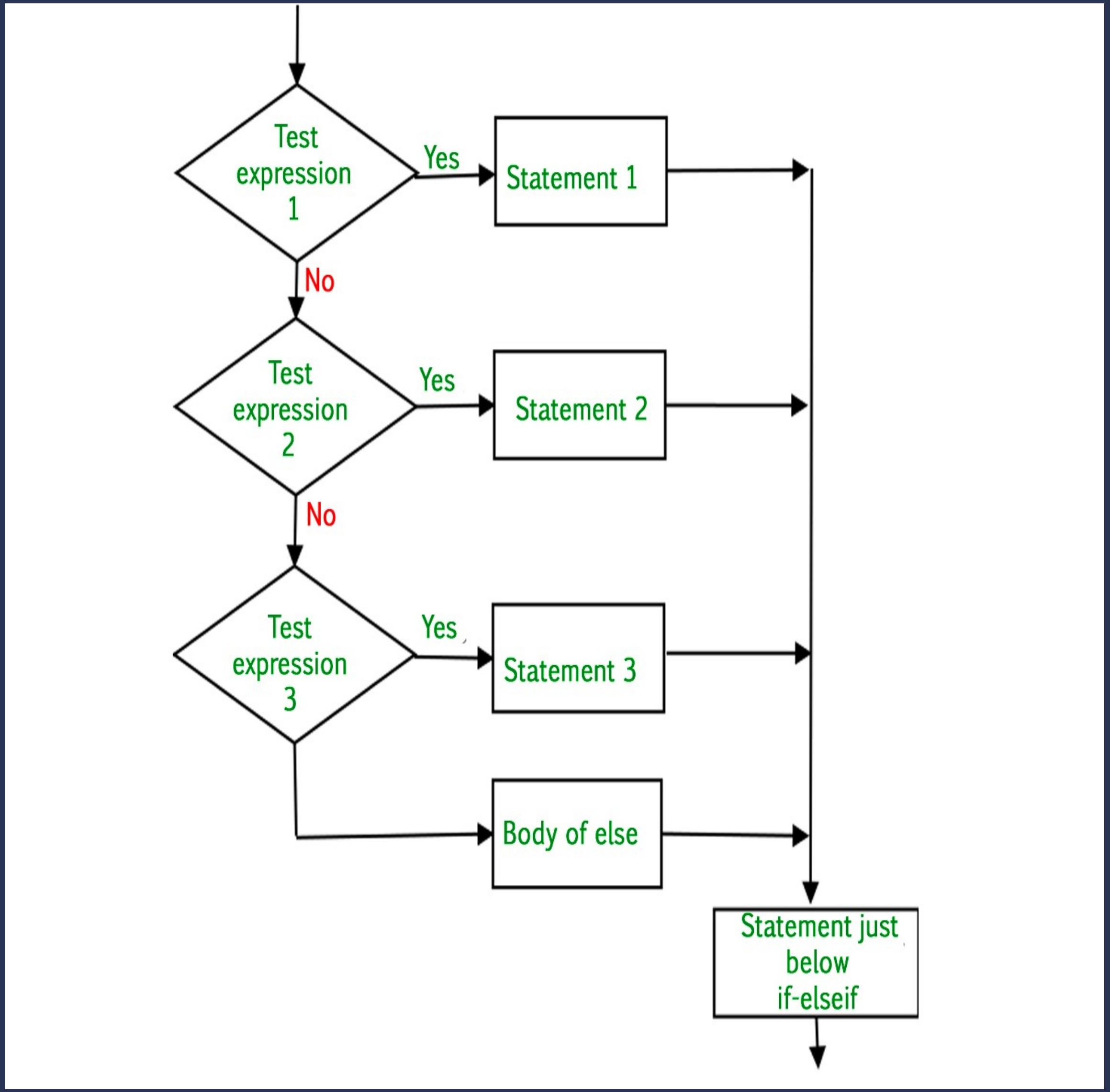


Decision making / Control Flow

if-elif-else ladder

Example:

```
i = 20
if (i == 10):
   print("i is 10")
elif (i == 15):
   print("i is 15")
elif (i == 20):
   print("i is 20")
else:
   print("i is not present")
```



Loops

Loops

A loop statement allows us to execute a statement or group of statements multiple times. There are following types of loops exists in python:

1. for loop: Executes multiple times a sequence of statements   that manages the loop variable.
2. while loop: Repeats a statement or group of statements while a given condition is TRUE. It tests the condition before executing the loop body.
3. nested loop: You can use one or more loop inside any another while, for loop.

Loop Control Statements

- Break statement: Terminates the loop statement and transfers execution to the statement immediately following the loop.
- Continue statement: Causes the loop to skip the remainder of its body and immediately retest its condition prior to reiterating.
- Pass statement: The pass statement in Python is used when a statement is required syntactically but you do not want any command or code to execute.

Refer Documentation: https://docs.python.org/3/tutorial/controlflow.html

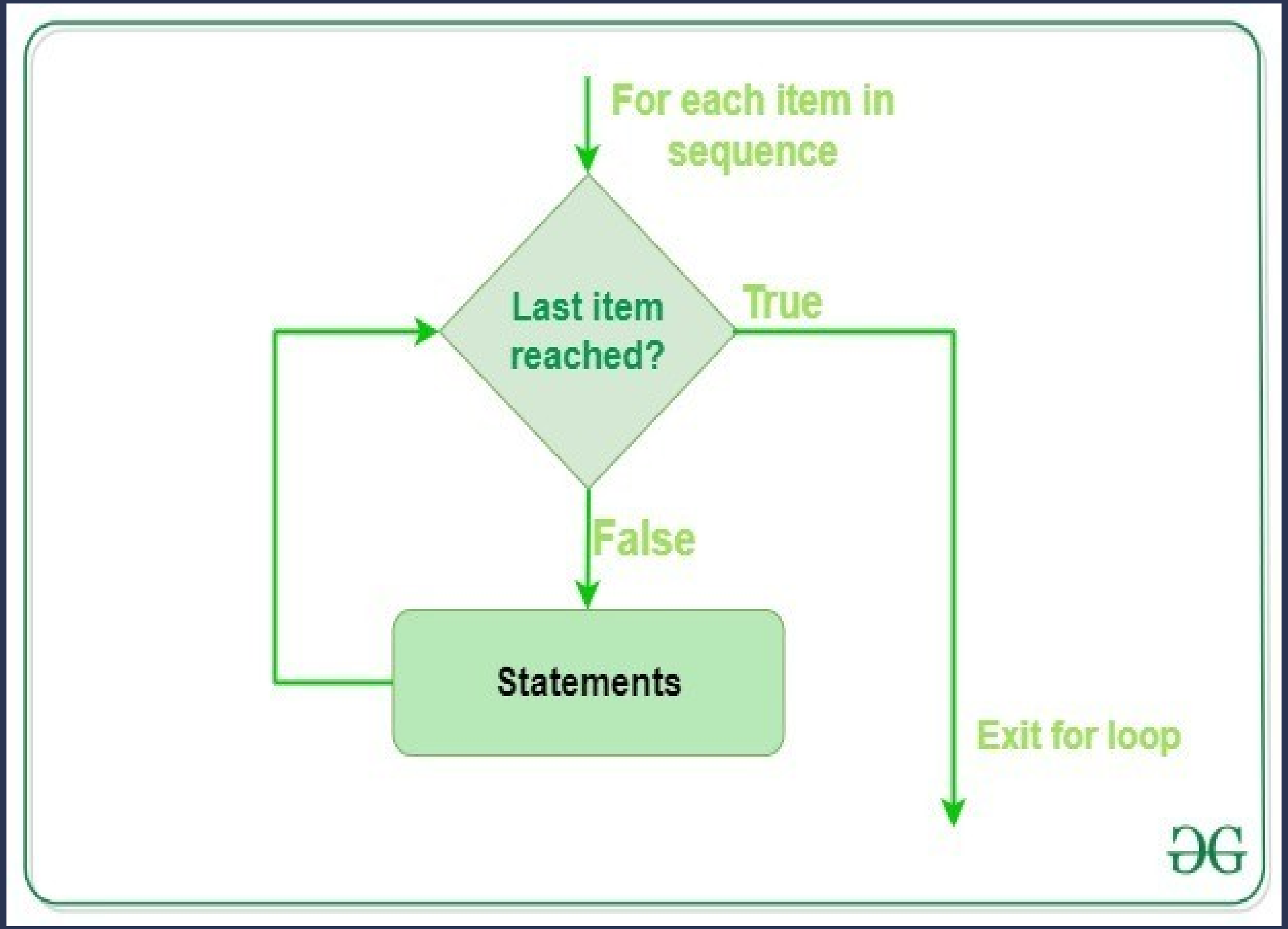
For Loop

Python For loop is used for sequential traversal i.e. it is used for iterating over an iterable like String, Tuple, List, Set, or Dictionary.

Refer Documentation: https://docs.python.org/3/tutorial/controlflow.html

Syntax:

```
for var in interable:

    #statements
```



For loop examples

- **For loop with list**

```
print(“List iteration”)
list1 = ["hello", "10",20, "end"]
print(“List iteration”)
for i in list1:
   print(i)
#Output = ?
```

- **For loop with string**

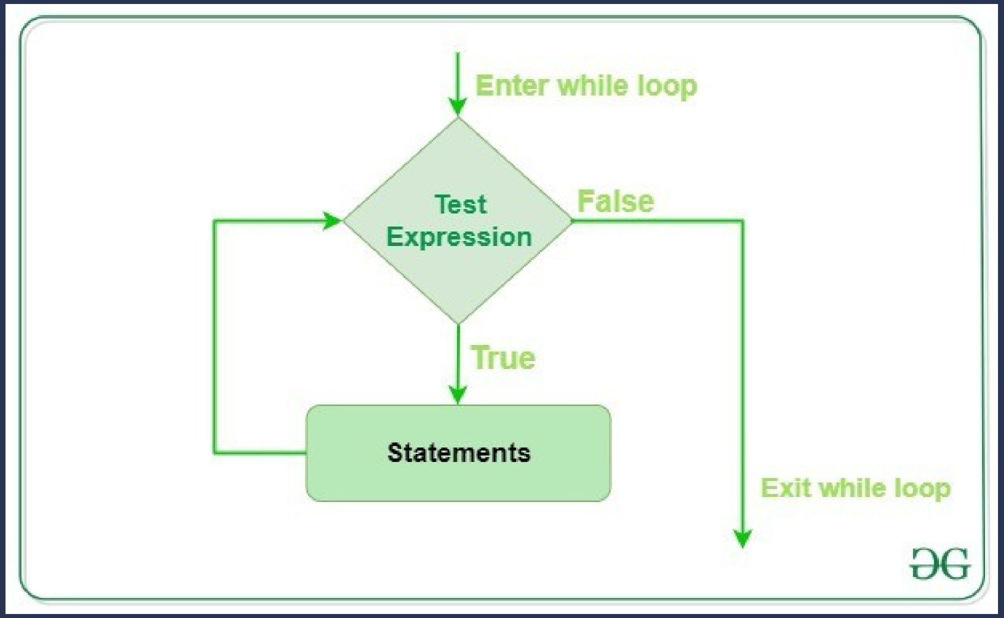
```
print(“String iteration”)
s = "India"
for i in s:
   print(i)
# Output =?
```

- **For loop with dictionary**

```
# Iterating over dictionary
print("Dictionary Iteration")
 d = {‘xyz’:123, ‘abc’:345}
for i in d:
   print("% s % d" % (i, d[i]))
#Output = ?
```

While loop

Python While Loop is used to execute a block of statements repeatedly until a given condition is satisfied. And when the condition becomes false, the line immediately after the loop in the program is executed.



While loop falls under the category of indefinite iteration. Indefinite iteration means that the number of times the loop is executed isn't specified explicitly in advance

- **Syntax**

```
while  expression:
           statement(s)
```

- **Example**

```
# Python program to illustrate while loop
count = 0
while (count < 4):
count = count + 1
print("Hello Geek")
#Output = ?
```

Nested loops

Python programming language allows to use one loop inside another loop.

<u>Syntax</u>

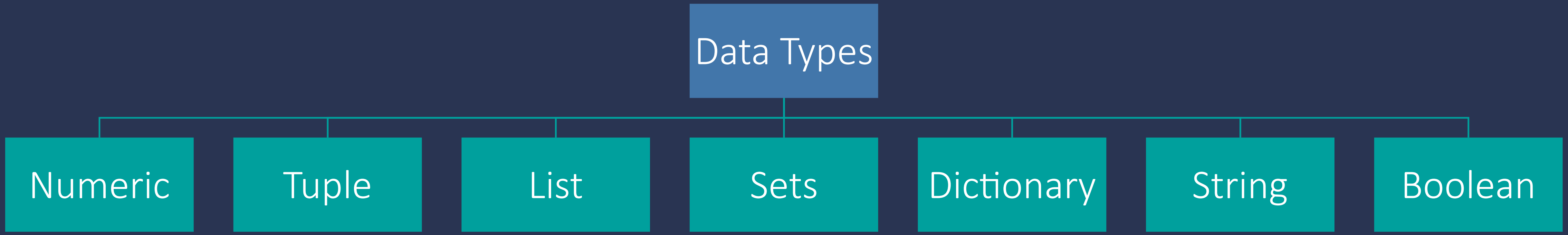
- **Nested for**

```
for iterating_var in sequence:
        for iterating_var in sequence:
                        statements(s)
Statements(s)
```

- **Nested while**

```
while expression:
        while expression:
                statement(s)
statement(s)
```

Operations on Data Types



List operations

1. Creation of list
2. Accessing element from list
3. Access subset of element using list slicing
4. Modifying list element
5. Adding elements in the existing list
6. Deleting an element
7. Concatenate 2 lists
8. Finding list length
9. Checking Membership of list element
10. To count occurrences of list element
11. Sort list elements in ascending order
12. Reverse the order of list elements

Dictionary operations

1. Creating Dictionary
2. Accesing dictionary elements
3. Modify dictionary elements
4. Adding new element: key-value pair
5. Deleting dictionary element
6. Membership checking
7. Getting keys and values
8. Copying dictionary
9. Removes all items from dictionary
10. Creating nested dictionary

Tuple operations

1. Tuple creation
2. Accessing tuple elements
3. Concatenation of 2 tuples
4. Repetition of tuple elements
5. Element Membership testing
6. To find Length of tuple
7. To unpack tuple elements
8. To count occurnces of a value in a tuple
9. To find the index of first occurence of a value
10. To convert tuple into list & vice versa
11. Usage of built-in function

Set operations

1. Creation of set
2. Adding and removing elements from a set
3. Set operations(union, intersection, diffrence, symmetric difference)

String operations

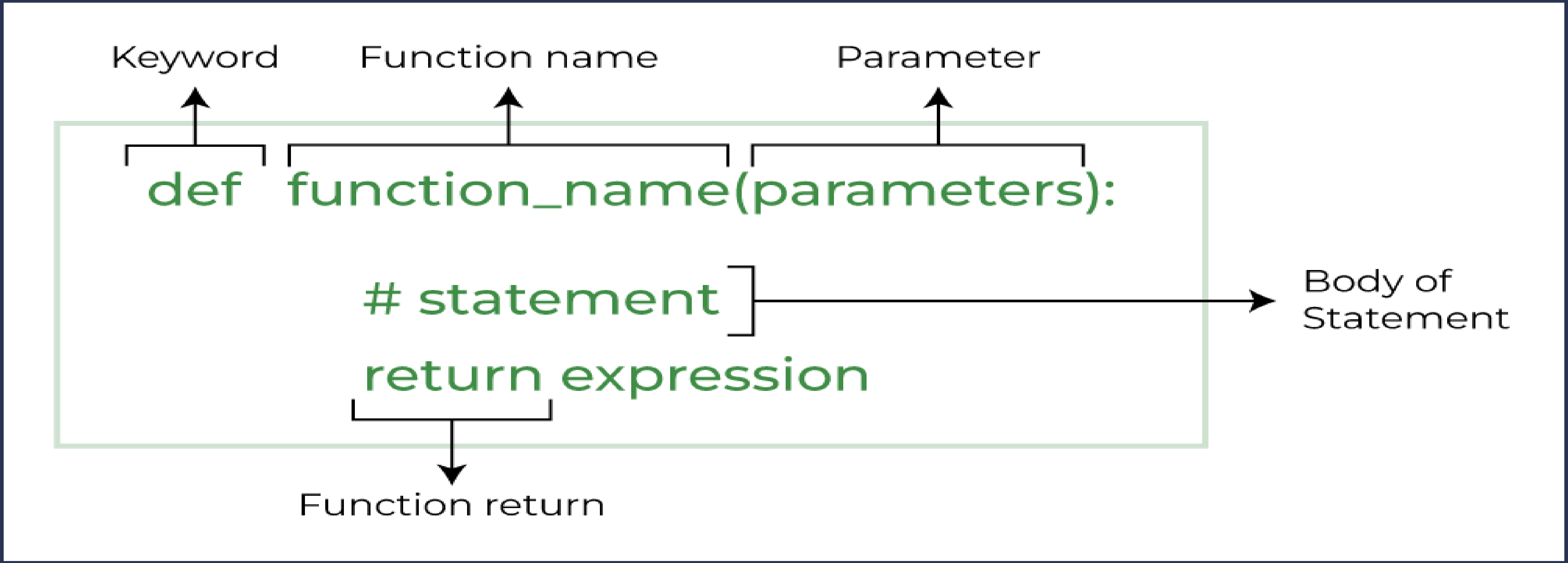
1. Creation of string
2. Accessing string elements
3. String concatenation
4. String repetition
5. To find string length
6. Iterating over string
7. Extracting substrings using slicing
8. Usage of different string methods
9. Formatting of a string
10. String comparision
11. Conversion of other data types to string
12. Extract raw string

Function, Modules and Packages in Python

Python has a wealth of useful functions, modules, and packages that can make programming a lot easier. In this presentation, we will go over some basics and examples that will help you master the art of Python programming!

Functions

- Refer documentation: https://docs.python.org/3/tutorial/controlflow.html#defining-functions
- Python Functions is a block of statements that return the specific task
- Functions increase code readability and code reusability



Modules

- Refer documentation: https://docs.python.org/3/tutorial/modules.html
- Module helps to organize python codes inside file.
- A module is a file consisting of Python code. A module can define functions, classes and variables.
- A module can also include runnable code.

Modules in Python

A module is a file containing Python definitions and statements. They can be used to organize code and make it easier to read. Here's more information:

1

What is a module?

A module is a Python file containing code for specific functionality.

2

Types of modules

There are built-in modules that come with Python and third-party modules that can be downloaded.

3

Importing modules

Use the 'import' statement followed by the module name to use a module in your code.

4

Example of using a module

Here is an example of using the 'math' module:

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

Packages in Python

Packages are collections of modules. They can be used to organize code into a hierarchical structure, making it even easier to read and understand. Here are some things you should know:

Understanding packages

A package is a folder that contains one or more modules or sub-packages.

Creating and organizing packages

To create a package, you need to create a folder with an '__init__.py' file in it. You can then organize modules and sub-packages within this folder.

Using packages in a project

You can then use the 'import' statement to import modules or sub-packages from your package folder.

Example of using a package

Here is an example of a package called 'my_package', containing the 'my_module' module:

```
import my_package.my_module
```