Introduction to python

Miss Muhammad Fatima Adam
Federal university Dutse

Google Research

Table of Contents!

01

Introduction

What is Python?

✔ Python…

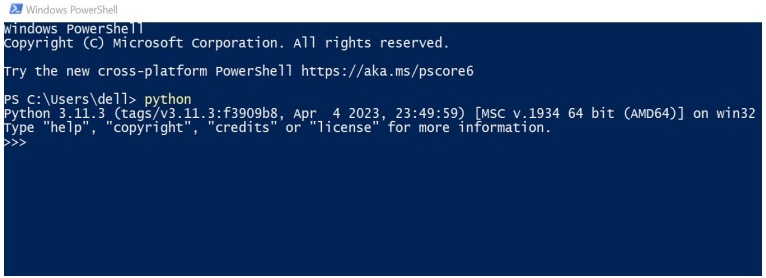
1. is a versatile and high-level programming language.
2. It is known for its simplicity and readability, making it an ideal choice for beginners.
3. Python is widely used in various fields, including web development, data science, and machine learning.
4. Python's extensive standard library and third-party packages make it a powerful tool for solving complex problems.
5. Throughout this course, we will explore the fundamentals of Python and how it is applied in the context of machine learning and data science.

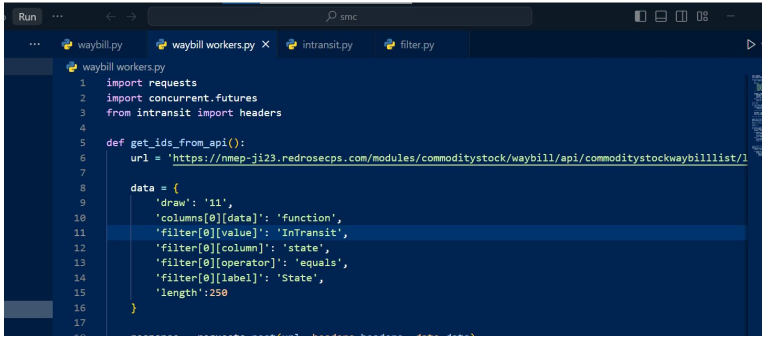
Setting up Python (IDEs, Jupyter Notebooks)

✔ **Download Python:** https://www.python.org/downloads/

1. Before we dive into Python programming, it's essential to set up your development environment.
2. IDEs (Integrated Development Environments) provide a user-friendly interface for writing and running Python code.
3. Jupyter Notebooks, on the other hand, offer an interactive and notebook-style environment, ideal for data exploration and documentation.
4. In this module, we will guide you through the process of installing Python and choosing the right development environment for your needs.

Ways to Run a Python





- **Interactive Mode:** You can run Python interactively by opening a terminal or command prompt and typing **python**.
- **Script Mode:** Create Python scripts by writing code in a .py file and running it using the python command followed by the script's filename.
- **Integrated Development Environments (IDEs):** IDEs like VS Code, and Jupyter Notebooks provide a comprehensive environment for writing, debugging, and running Python code efficiently.

Writing and executing your first Python program using Colab

- **What is Colab?**
- Colab, or "Colaboratory", allows you to write and execute Python in your browser, with:
- Zero configuration required
- Access to GPUs free of charge
- Easy sharing

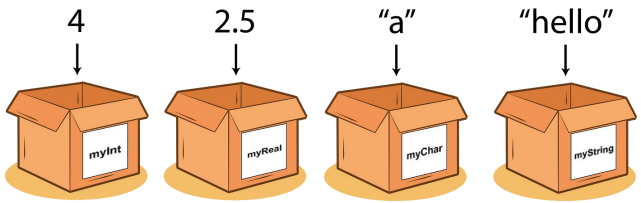
```python
#This is comment
#We use comment to make our code readable

#A variable declaration
spam_amount = 0

#calling print function
print(spam_amount)

# Ordering Spam, egg, Spam, Spam, bacon and Spam
spam_amount = spam_amount + 4
```

02

Variables, Operators, and Data Structures

Variables

- In Python, variables are used to store data values.

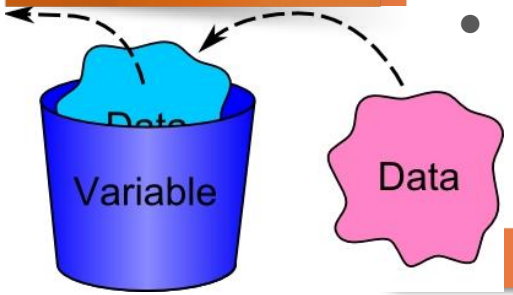


- Variables are created when you assign a value to them.
- Python has dynamic typing, meaning you don't need to declare a variable's type explicitly.

- Python supports various data types, including:
  - **int**: Integers (e.g., 5, -10)
  - **float**: Floating-point numbers (e.g., 3.14, -0.5)
  - **str**: Strings (e.g., "Hello, World!")
  - **bool**: Boolean values (True, False)

More on Variables

- In Python, variables are used to store data values.
- Variables are created when you assign a value to them.
- Python has dynamic typing, meaning you don't need to declare a variable's type explicitly.



Creating Variables

In Python, you can create variables by assigning a value to them using the assignment operator (=).

Variable_Name = variable_data

Variable names must follow certain rules:

- They can contain letters, numbers, and underscores.
- They cannot start with a number.
- Variable names are case-sensitive (e.g., myVar and myvar are different variables).
- Variables can be reassigned at any time

```
In [1]: a=1

In [2]: b=2

In [3]: a
Out[3]: 1

In [4]: b
Out[4]: 2

In [5]: a=b

In [6]: a
Out[6]: 2

In [7]: b=-0.15
```

Arithmetic Operators

Operator	Description
+	addition
–	subtraction
*	multiplication
/	division
**	exponentiation
//	integer division
%	modulo (remainder)

Python supports a wide variety of operators which act like functions, i.e.
they do something and return a value:

Comparison Operators

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

03

Data Structures

Data Structures

Data structures are used to store and organize multiple pieces of data.

Python provides several built-in data structures

List

Ordered, mutable collections of data.

Tuples

Ordered, immutable collections of data.

Dictionaries

Unordered collections of key-value pairs.

[List]

- Lists are one of the most commonly used data structures in Python.
- They are ordered and mutable, which means you can change their contents.
- Lists can contain elements of different data types.
- Lists can be created by enclosing elements in square brackets [ ].
- For example:
- my_list = [1, 2, 3, 4, 5]
- You can access individual elements in a list using indexing, with the first element at index 0.
- For example:
- first_element = my_list[0]

Common list operations include:

- Appending elements with .append().
- Example:

my_list.append(6)

- Removing elements with .remove() or .pop().
- Example:

my_list.remove(3)

(Tuples)

- Tuples are similar to lists but with a key difference: they are immutable.
- Once you define a tuple, you cannot change its elements.
- Tuples are often used for data that should not be modified.
- Tuples are created by enclosing elements in parentheses ( ).
- For example:

```
my_tuple = (1, 2, 3, 4, 5)
```

- You can access individual elements in a tuple using indexing, just like with lists.
- For example:

```
first_element = my_tuple[0]
```

Dictionaries

Dictionaries are collections of key value pairs.

They are unordered and mutable, allowing for flexible data storage.

Each element in a dictionary is accessed by its key.

Dictionaries are commonly used to store and retrieve data associated with specific identifiers.

We'll explore how to create and work with dictionaries in Python..

Creating and Using Dictionaries

Dictionaries are created by enclosing key-value pairs in curly braces {}
For example:

```
my_dict = {"name": "John", "age": 30,
           "city": "New York"}
```

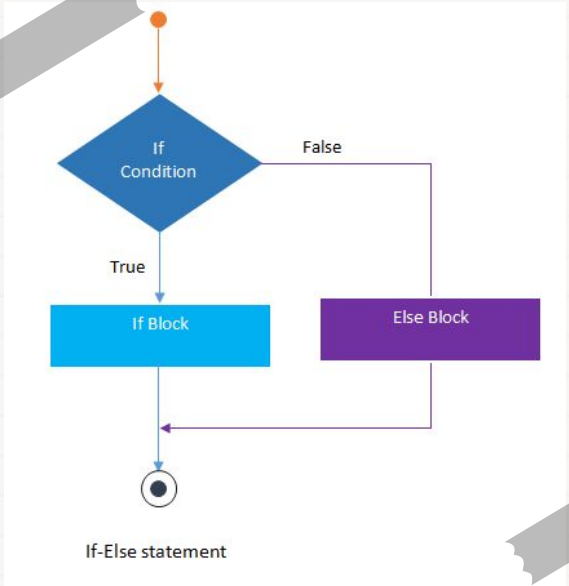
You can access values in a dictionary by specifying the key.

```
name = my_dict["name"]
```

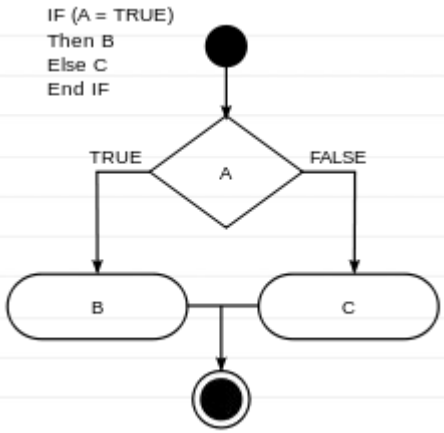
04

Control Structures

learn about conditional statements, looping, and how to control the flow of your Python programs.



Conditional Statements in Python (if, elif, else)



- Conditional statements allow you to make decisions in your Python programs.

- In Python, you can use if, elif (short for "else if"), and else to control the flow of your code based on conditions.

- The if statement allows you to execute a block of code if a condition is true.
- It's the fundamental building block of conditional logic in Python.

Using "if" Statements

```
if condition:
    # Code to execute if the condition is true
```

Example:

python

```python
temperature = 25
if temperature > 30:
    print("It's a hot day!")
```

"elif" and "else" Statements

"elif" (else if) and "else" statements for more complex conditional logic.

"elif" allows you to check multiple conditions sequentially

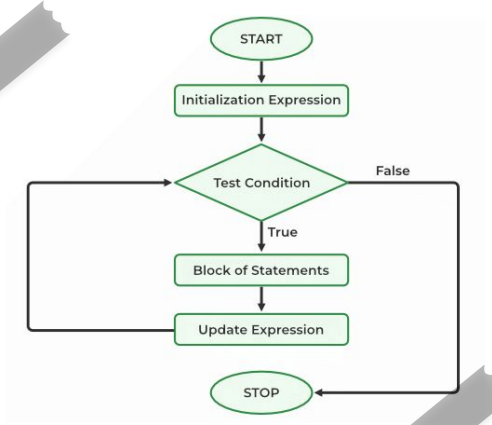
"else" provides a fallback option when no previous condition is true.

```python
if condition1:
    # Code to execute if condition1 is true
elif condition2:
    # Code to execute if condition2 is true
else:
    # Code to execute if no previous condition is true
```

Exampl
e

```python
age = 18
if age < 18:
    print("You are a minor.")
elif age >= 18 and age < 65:
    print("You are an adult.")
else:
    print("You are a senior citizen.")
```

Loops



loop is a sequence of instruction s that is continually repeated until a certain condition is reached.

- Loops allow you to execute a block of code repeatedly.
- Python supports two primary types of loops: "for" loops and "while" loops.
- "for" loops are used for iterating over sequences (e.g., lists, strings).
- "while" loops continue execution as long as a specified condition is true.
- We'll explore how to use these loops effectively.

For Loop

Syntax:

```python
for item in sequence:
    # Code to execute for each item
```

Example:

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

While Loop

Syntax:

```python
while condition:
    # Code to execute as long as the condition is true
```

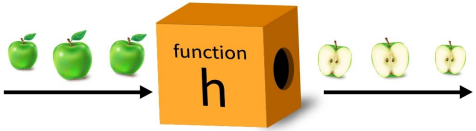
Example:

```python
count = 0
while count < 5:
    print("Count:", count)
    count += 1
```

05

Functions and Modules

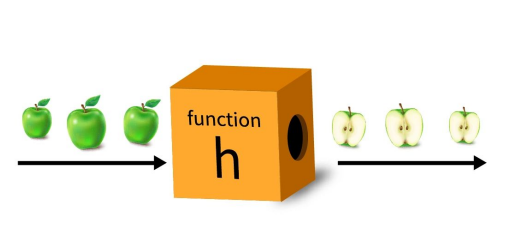
Building reusable code blocks, delve into built-in Python functions and importing external modules.



Functions

Functions are blocks of reusable code that perform specific tasks.

They allow you to organize your code, make it more readable and avoid repetition.



To create a function in Python, use the **def** keyword, followed by the function name and parentheses.

Syntax:

```python
def function_name(parameters):
    # Function body
    # Code to execute
```

Example:

```python
def greet(name):
    return "Hello, " + name + "!"

message = greet("Alice")
print(message)
```

Built-in Functions

Python provides a wealth of built-in functions that perform common tasks.

These functions can be used without the need for defining them.

Examples of built-in functions include:

- **print()**: Used to display output.
- **len()**: Returns the length of a sequence (e.g., a list or a string).
- **sum()**: return summation of list of tuple.

Example:

```python
name = "Alice"
print("Name:", name)
length = len(name)
print("Length of name:", length)
```

Example:

```python
numbers = [1, 2, 3, 4, 5]
sum_of_numbers = sum(numbers)
print("Sum of numbers:", sum_of_numbers)
```

Importing and Using Modules

Python modules are collections of related functions and code that can be reused in different programs.

You can import modules using the **import** statement.

Importing modules allows you to access additional functionality not available in Python's built-in functions.

We can see all the names in module using the built-in function dir().

Example:

```python
import math
result = math.sqrt(25)
```

05

NumPy and Arrays

A fundamental library for numerical operations

NumPy

NumPy

NumPy, short for Numerical Python, is a powerful Python library for numerical and array-based operations.

It provides support for large, multi-dimensional arrays and matrices, along with a collection of mathematical functions.

NumPy is a fundamental library for data science, machine learning, and scientific computing.

Numpy ndarray for creating multiple dimensional arrays
NumPy-based algorithms are generally 10 to 100 times faster

We'll explore its capabilities and how to get started with NumPy in our interactive python notebook.

NDarray

- NumPy adds a new data structure to Python(ndarray)
  - An N-dimensional array is multidimensional container of items of the same type and size.Defined by:

1. the shape of the array, and
2. the kind of item the array is composed of
3. The shape of the array is a tuple of N integers (one for each dimension)

python

```python
import numpy as np
arr = np.array([1, 2, 3, 4, 5])
```

```python
import numpy as np

A = np.array([[1, 2, 3], [4, 5, 6]])
print A
# [[1 2 3]
#  [4 5 6]]
```

Basic Operations with NumPy Arrays

NumPy arrays support a wide range of mathematical and element-wise operations.

You can perform operations such as addition, subtraction, multiplication, and more on arrays.

NumPy simplifies complex numerical computations.

Example:

python

```python
import numpy as np
arr1 = np.array([1, 2, 3])
arr2 = np.array([4, 5, 6])
result = arr1 + arr2
print("Array Addition:")
print(result)
```

NumPy documentation

Official documentation
http://docs.scipy.org/doc/

The NumPy book
http://www.tramy.us/numpybook.pdf

Example list
http://www.scipy.org/Numpy_Example_List_With_Doc

python

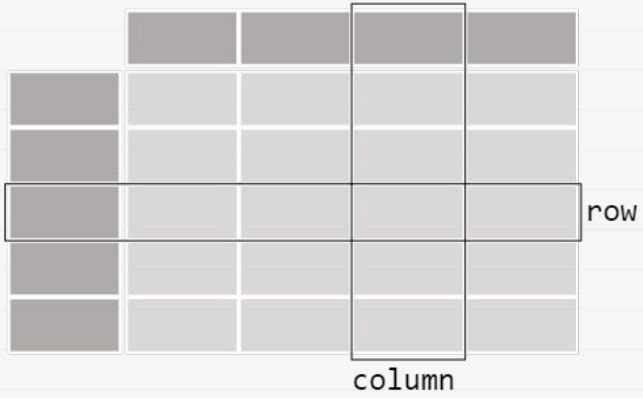
```python
import numpy as np

# Example: Calculating Mean and Median
data = np.array([12, 18, 24, 30, 36, 42, 48, 54, 60])
mean = np.mean(data)
median = np.median(data)

print("Data:", data)
print("Mean:", mean)  # Output: Mean: 36.0
print("Median:", median)  # Output: Median: 36.0
```

05

Panda

a Python library that provides easy-to-use data structures and data analysis tools.

pandas



Series & DataFrames

The primary two components of pandas are the Series and DataFrame.

Series is essentially a column, and

DataFrame is a multi-dimensional table made up of a collection of Series.

Series

	apples
0	3
1	2
2	0
3	1

+

Series

	oranges
0	0
1	3
2	7
3	2

=

DataFrame

	apples	oranges
0	3	0
1	2	3
2	0	7
3	1	2

Creating a DataFrame from scratch

- There are many ways to create a DataFrame from scratch, but a great option is to just use a simple dict. But first you must import pandas.

```
import pandas as pd
```

- Let's say we have a fruit stand that sells apples and oranges. We want to have a column for each fruit and a row for each customer purchase. To organize this as a dictionary for pandas we could do something like:

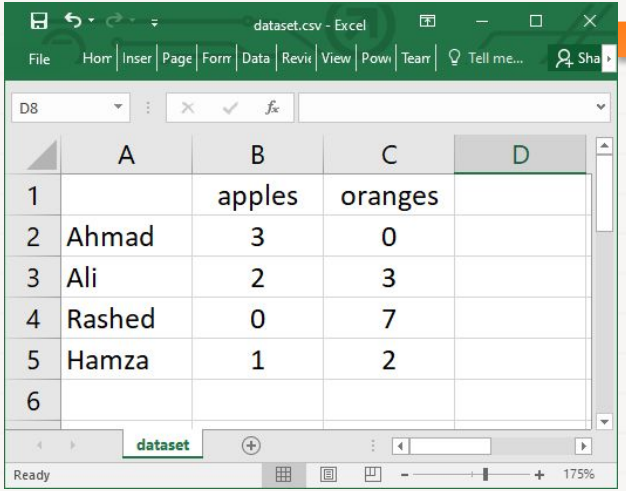
```
data = { 'apples':[3, 2, 0, 1] , 'oranges':[0, 3, 7, 2] }
```

- And then pass it to the pandas DataFrame:

```
df = pd.DataFrame(data)
```

	apples	oranges
0	3	0
1	2	3
2	0	7
3	1	2

Reading data from a CSV file



	A	B	C	D
1		apples	oranges	
2	Ahmad	3	0	
3	Ali	2	3	
4	Rashed	0	7	
5	Hamza	1	2	
6				

Example:

python

```python
import pandas as pd

# Reading a CSV file
df = pd.read_csv('data.csv')
```

Viewing your data

- The first thing to do when opening a new dataset is print out a few rows to keep as a visual reference. We accomplish this with `.head()`:

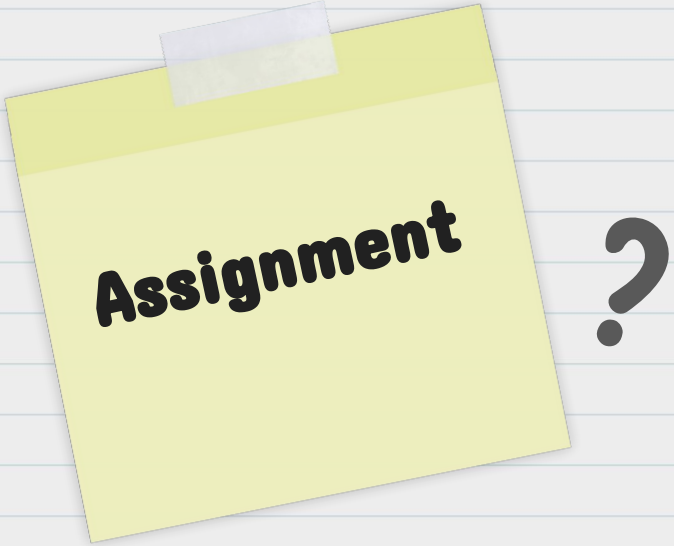
```
df.head()
```

- .head() outputs the first five rows of your DataFrame by default, but we could also pass a number as well: `df.head(10)` would output the top ten rows, for example.

- To see the last five rows use `.tail()` that also accepts a number, and in this case we printing the bottom two rows.:

```
df.tail(2)
```

Lab Session



?

Thanks

slidesgo