DM,
Autumn, 2026

Overview of Python & Its Role in Bussiness Anaytics

Faculty of DS & AI
Autumn semester, 2026

Trong-Nghia Nguyen

Business AI Lab

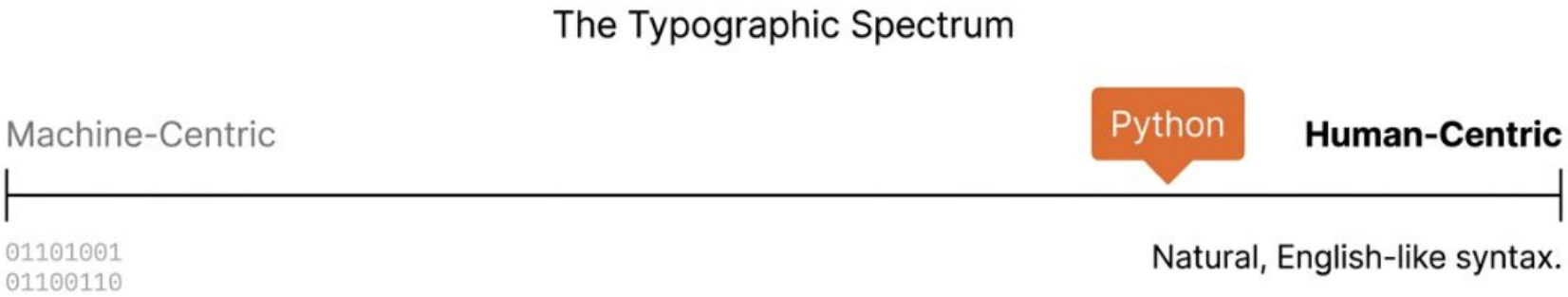
Content

- **Overview**
- Foundation
- Numpy
- Function

Overview

Python

Python is designed for human readability rather than machine efficiency



A high-level programming language that prioritizes ease of reading and writing.

The natural syntax reduces the cognitive load of coding, aligning perfectly with the business analyst's need to focus on logic and strategy rather than complex syntax.

Overview

Python

NumPy

Core Function:
Numerical Python.

Application:
Specialized in high-performance calculations for numerical arrays.

pandas

Core Function:
Tabular Data.

Application:
Engineered for data manipulation and analysis (functions similarly to an advanced Excel).

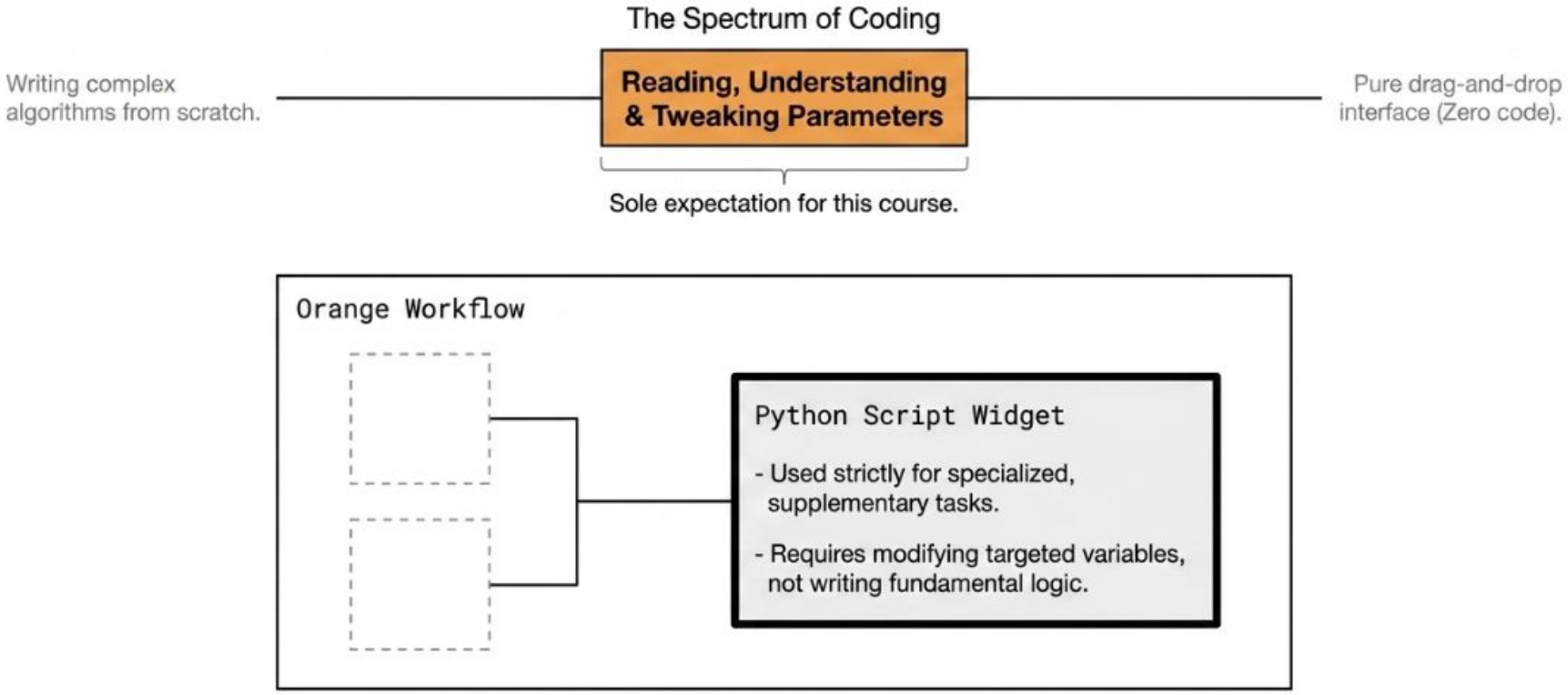
scikit-learn

Core Function:
Machine Learning.

Application:
Provides robust, standard algorithms for predictive modeling and data mining.

Overview

Python with Orange



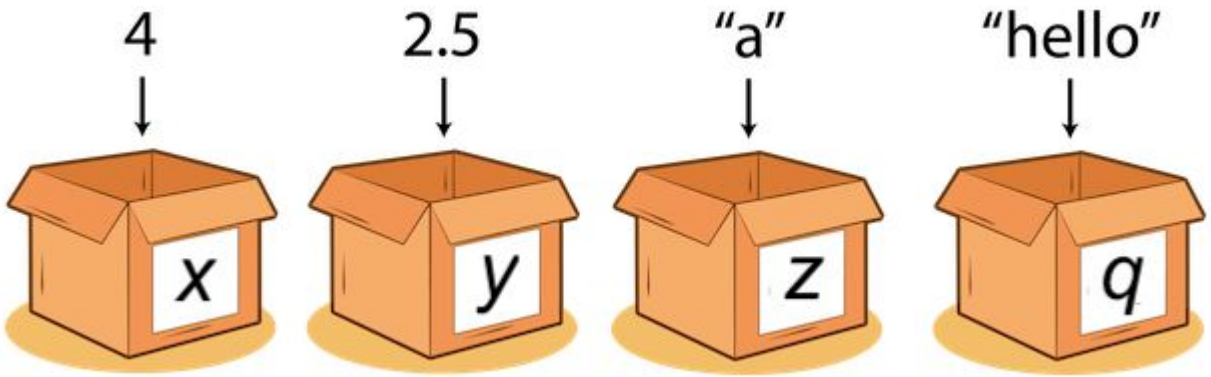
Content

- Overview
- **Foundation**
- Numpy
- Function

Foundation

Variable & Type

- A **variable** is a name that represents a **value**.
- Think of it as a box containing information.
- We use the assignment operator `=` to store a value in a variable.
- The `type()` function reveals the data **type**, while `print()` outputs it.



Variable Types	Basic Examples	Real World Examples
Integer	Z = 1	Release_Year= 2003
Float	Z = 1.23	Rating = 8.2
String	Z = "Python"	Movie = "Finding Nemo"
Boolean	Z = True	Is_Released = True
List	Z = ["Alice", "Lily"]	Actors = ["Elen DeGeneres", "Alexander Gould"]
Dictionary	Z = {"Name": "Alan", "ID": 1}	Box_Office_Collection= {"US": "339.7M", "UK": "67.1M"}
Tuple	Z= ("Monday", 26, "June")	Release_Date = ("30th", "May", 2003)
Set	Z= {2,3,5}	Genre = {"Animation", "Family", "Adventure"}

Foundation

Variable & Type

- A **variable** is a name that represents a **value**.
- Think of it as a box containing information.
- We use the assignment operator `=` to store a value in a variable.
- The `type()` function reveals the data **type**, while `print()` outputs it.

Variable Types	Basic Examples	Real World Examples
Integer	Z = 1	Release_Year= 2003
Float	Z = 1.23	Rating = 8.2
String	Z = "Python"	Movie = "Finding Nemo"
Boolean	Z = True	Is_Released = True
List	Z = ["Alice", "Lily"]	Actors = ["Elen DeGeneres", "Alexander Gould"]
Dictionary	Z = {"Name": "Alan", "ID": 1}	Box_Office_Collection= {"US": "339.7M", "UK": "67.1M"}
Tuple	Z= ("Monday", 26, "June")	Release_Date = ("30th", "May", 2003)
Set	Z= {2,3,5}	Genre = {"Animation", "Family", "Adventure"}

```
# Assigning a string to a variable
course_name = "Data Mining"

# Checking the data type and printing the output
print(type(course_name))  # Output: <class 'str'>
```

Foundation

Arithmetic operator

Main.py

```
1  a = 1 + 2 + 3 + 4 + 5  # cộng
2  b = 2 * 3.14159  # nhân
3  c = 2 ** 10  # hàm mũ
4
5  print(a)
6  print(b)
7  print(c)
```

Operators	Meaning	Example	Result
+	Addition	4 + 2	6
–	Subtraction	4 – 2	2
*	Multiplication	4 * 2	8
/	Division	4 / 2	2
%	Modulus operator to get remainder in integer division	5 % 2	1
**	Exponent	5**2 = 5^2	25
//	Integer Division/ Floor Division	5//2 -5//2	2 -3

Main.py

Chạy mã | Đầu vào | Lưu bài | Tải file

```
1  int_2 = 2                    # Gán giá trị 2 cho biến int_2 (kiểu số nguyên - integer)
2  print(type(int_2))           # Kiểm tra kiểu dữ liệu của biến int_2 (kết quả: <class 'int'>)
3  int_2 / int_2                # Thực hiện phép chia 2 chia cho 2 (kết quả: 1.0)
4  print(type(int_2 / int_2))   # Kiểm tra kiểu dữ liệu của kết quả phép chia (kết quả: <class 'float'>)
```

Foundation

Arithmetic operator

Danh sách bài nộp

Main.py +

```
1  print(101 / 2)
```

Danh sách bài nộp

Main.py +

```
1  print(101 // 2)  # phép chia làm tròn xuống
```

Operators	Meaning	Example	Result
+	Addition	4 + 2	6
–	Subtraction	4 – 2	2
*	Multiplication	4 * 2	8
/	Division	4 / 2	2
%	Modulus operator to get remainder in integer division	5 % 2	1
**	Exponent	5**2 = 5^2	25
//	Integer Division/ Floor Division	5//2 -5//2	2 -3

Danh sách bài nộp

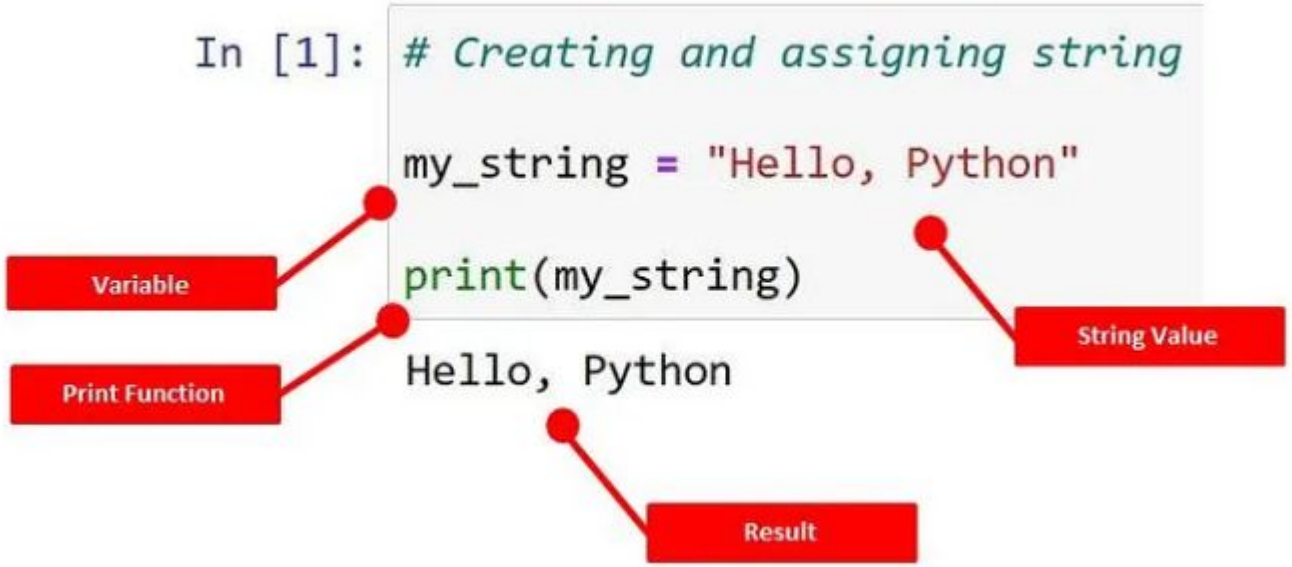
Main.py +

```
1  print(101 % 2)  # 101 mod 2
```

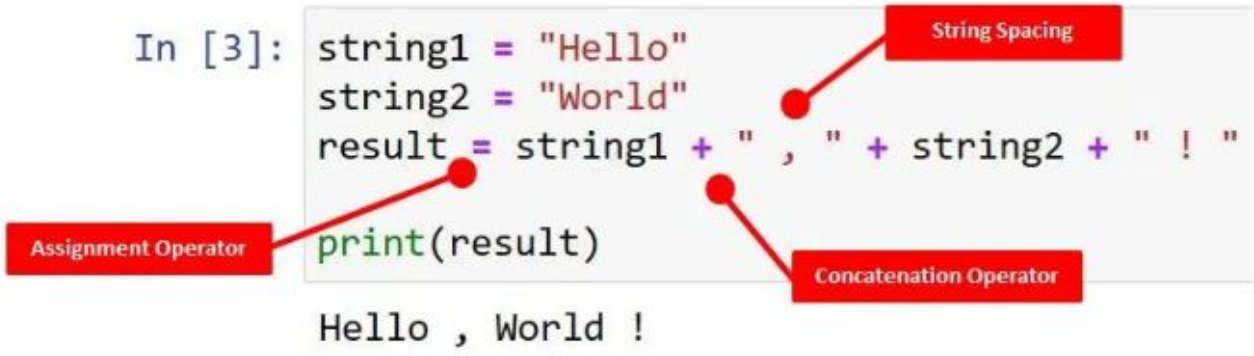
Foundation

String

- Characters, character sequences, or text are stored as a variable type known as a **string**.
- In many programming languages, a **string** is also referred to as a sequence of characters.



String concat:



Foundation

String

- Characters, character sequences, or text are stored as a variable type known as a **string**.
- In many programming languages, a **string** is also referred to as a sequence of characters.

```
In [1]: # Creating and assigning string

        my_string = "Hello, Python"

        print(my_string)

        Hello, Python
```

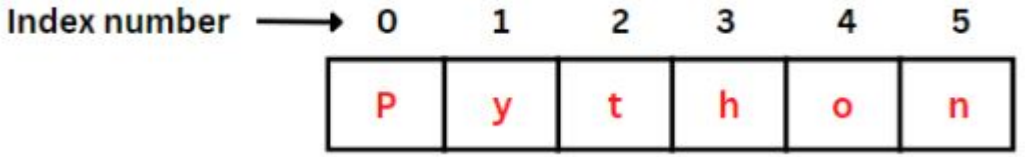
Variable

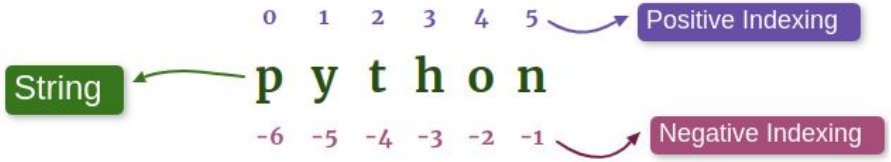
Print Function

String Value

Result

String index:





Foundation

String

String index:

Ex 1:

```
# declaring the string
s = "Geeks for Geeks !"

# accessing the character of str at 0th index
print(s[0])

# accessing the character of str at 6th index
print(s[6])

# accessing the character of str at 10th index
print(s[10])
```

Ex 2:

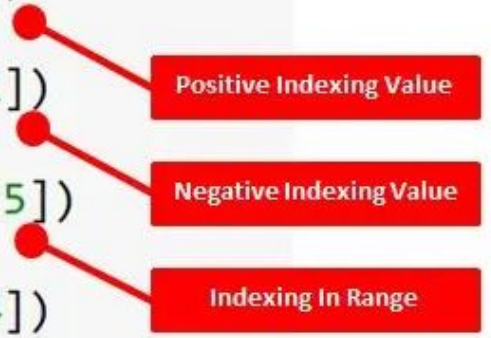
```
# declaring the string
s = "Geeks for Geeks !"

# accessing the character of str at last index
print(s[-1])

# accessing the character of str at 5th index from the last
print(s[-5])

# accessing the character of str at 10th index from the last
print(s[-10])
```

Foundation

String

String slicing:

```
In [7]: # String indexing and slicing

        my_string = "Hello , World"
        print(my_string[0])

        print(my_string[-1])

        print(my_string[2:5])

        print(my_string[:4])

        print(my_string[2:])
```




```
H
d
llo
Hell
llo , World
```

Foundation

List & Tuple

Lists: Ordered & Mutable

Lists can be modified

Lists can be modified (items added, removed, or changed) after creation.

```
# Creating a mutable List using
square brackets
tools = ['Python', 'R']

# Adding a new element
tools.append('SQL')

# Checking the number of elements
print(len(tools)) # Output: 3
```

Tuples: Ordered & Immutable

Tuples are locked. They represent fixed data that cannot be altered after creation.

```
# Creating an immutable Tuple
using parentheses
fixed_tools = ('Python', 'R')

# tools.append('SQL') would
cause an AttributeError
```

Foundation

Boolean

- A logical variable—referred to in Python as the `**bool**` type—is a variable that holds only two possible values: `**True**` and `**False**`.

```
a = 1
a == 1
```
```
True
```
```
a != 10
```
```
True
```
```
a != 1
```
```
False
```
```
a > 10
```
```
False
```
```
a < 12
```
```
True
```
```
a >= 1
```
```
True
```
```
a <= 7
```
```
True
```

```
bool_var = True
bool_var
```
```
True
```

Operator	Meaning
== (double equal to)	Equal to
<	Less than
>	Greater than
!=	Not equal to
<=	Less than or equal to
>=	Greater than or equal to

Foundation

Boolean

- A logical variable—referred to in Python as the `**bool**` type—is a variable that holds only two possible values: `**True**` and `**False**`.
- For Operator, we have: '**AND**'; '**OR**'; '**NOT**'.

X	Y	X and Y	X or Y	not(X)	not(Y)
T	T	T	T	F	F
T	F	F	T	F	T
F	T	F	T	T	F
F	F	F	F	T	T

Dictionary

Dictionaries: Key-Value Mapping

Dictionaries bypass numerical order, mapping unique **Keys** to specific **Values** for direct data retrieval.

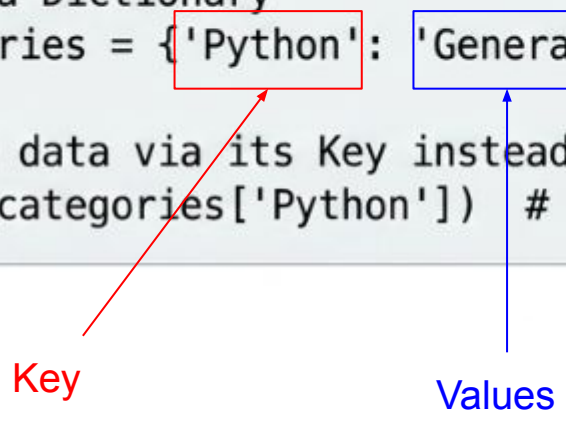
```
# Creating a Dictionary
tool_categories = {'Python': 'General', 'SQL': 'Database'}

# Accessing data via its Key instead of position
print(tool_categories['Python'])  # Output: General
```

Dictionary

Dictionaries: Key-Value Mapping

Dictionaries bypass numerical order, mapping unique **Keys** to specific **Values** for direct data retrieval.

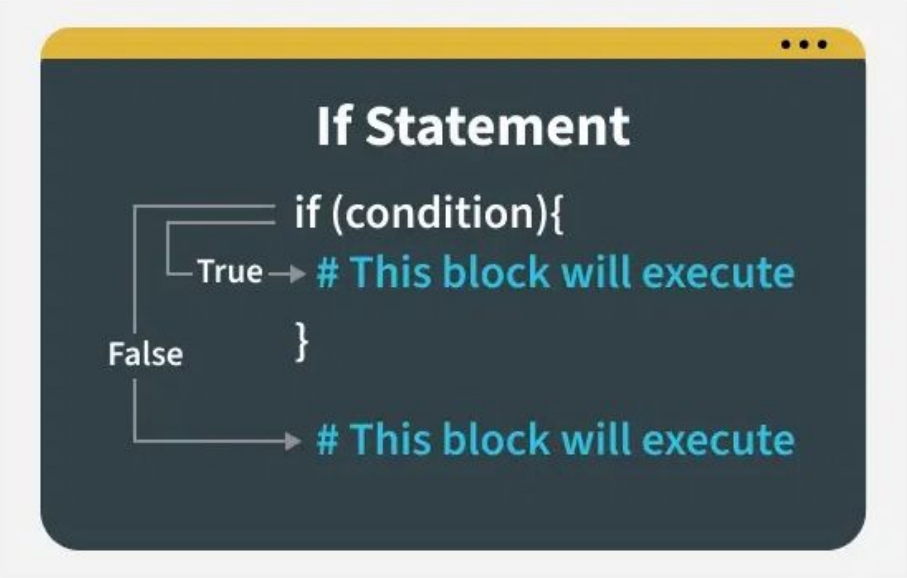
```
# Creating a Dictionary
tool_categories = {'Python': 'General', 'SQL': 'Database'}

# Accessing data via its Key instead of position
print(tool_categories['Python'])  # Output: General
```



Foundation

IF/else

- The **if** statement is used to execute a block of code only when a given condition is True.
- If the condition is False, the code inside the if block is skipped.



If Statement

```
i = 10

if i > 15:
    print("10 is less than 15")

print("I am Not in if")
```

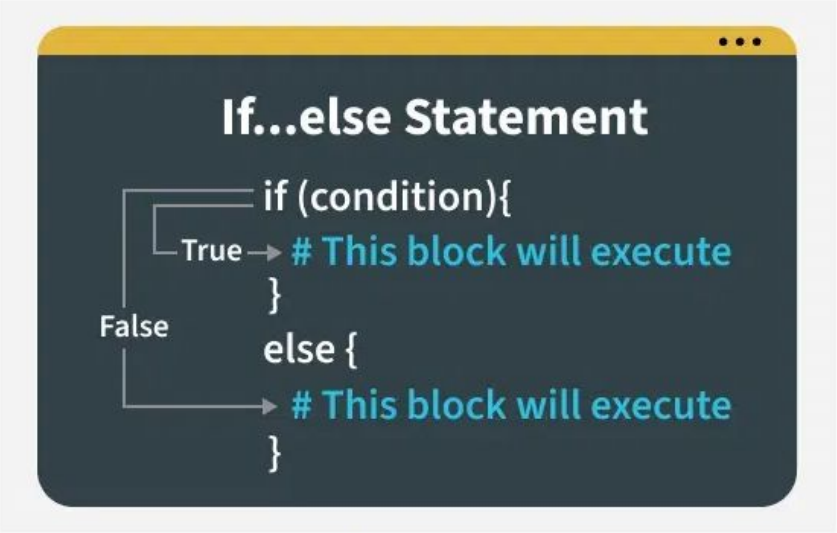
Output

```
I am Not in if
```

Foundation

IF/else

- The **if-else** statement is used to execute one block of code when a condition is True and another block when the condition is False.
- It helps programs make decisions based on different conditions.



If-else Statement

```
i = 20

if i > 0:
    print("i is positive")
else:
    print("i is 0 or Negative")
```

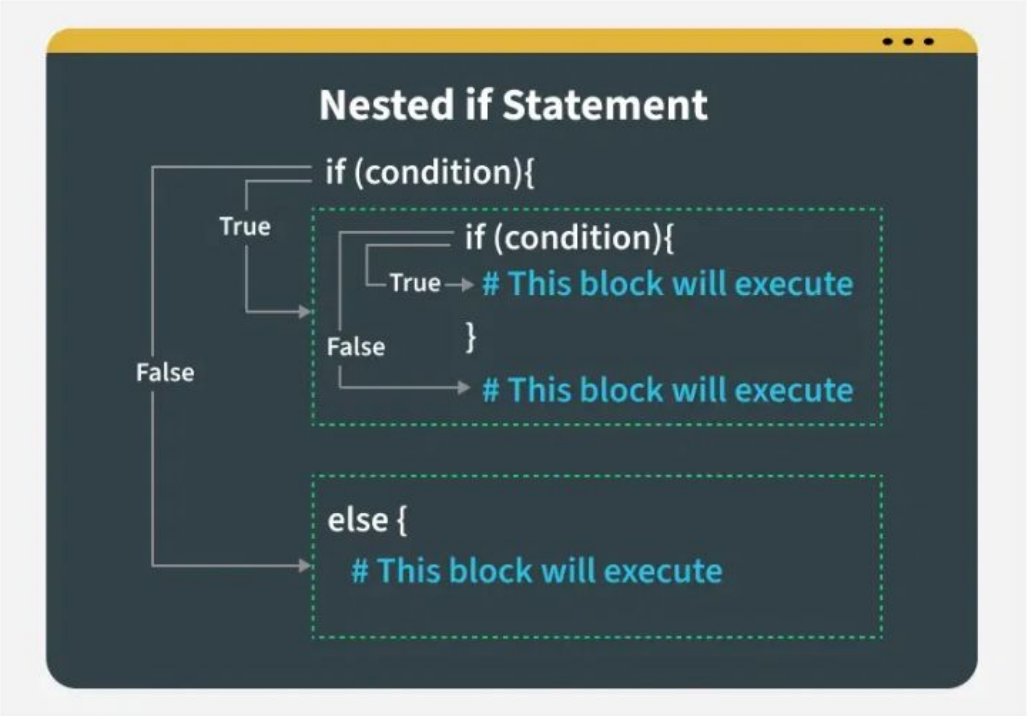
Output

```
i is positive
```

Foundation

Nested If-Else Statement

- A nested if-else statement is an if-else structure placed inside another if or else block.
- It is used to check multiple conditions step by step and execute code based on those conditions.



Nested if Statement

```
i = 10
if i == 10:

    if i < 15:
        print("i is smaller than 15")

    if i < 12:
        print("i is smaller than 12 too")
    else:
        print("i is greater than 15")

else:
  print("i is not equal to 10")
```

Foundation

For loop

- Python **for** loops are used to iterate over sequences such as lists, tuples, strings, and ranges.
  - Allows the same operation to be applied to every item in a sequence.
  - Avoids the need to manage loop indices manually.

```
a = ["Geeks", "for", "Geeks"]
for i in a:
    print(i)
```

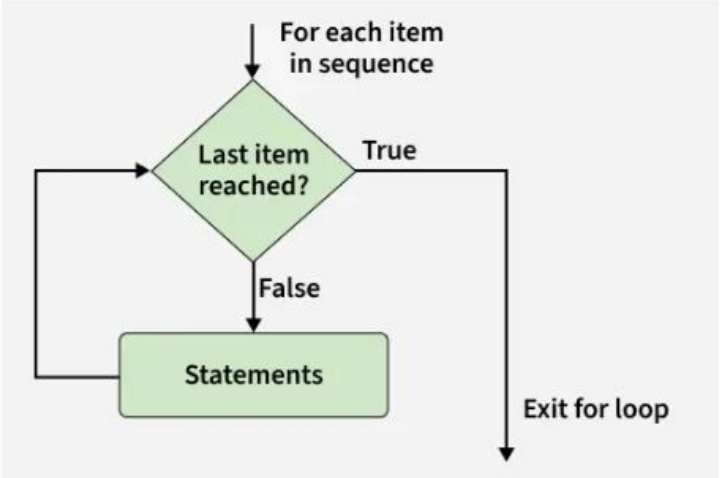
Output

```
Geeks
for
Geeks
```

Explanation:

- a stores a list of strings.
- for i in a: iterates through each item in the list.

Flowchart of For Loop



Flowchart of For Loop

Indexing in a List

- Similar to string
- Python uses zero-based indexing.
- The first element is [0], the second element is [1], and so on.
- Negative index counting backwards from the end

```
tools = ['Python', 'R', 'Java', 'SQL']

tools[0]    # 'Python'  (phần tử đầu)

tools[2]    # 'Java'
```

```
tools[-1]   # 'SQL'   (phần tử cuối)

tools[-2]   # 'Java'
```

Foundation

Slicing in a List

- Similar to string
- Syntax: **x[start:stop:step]** — extracts elements from start up to, but not including, stop.

```
nums = [10, 20, 30, 40, 50]

nums[1:3]    # [20, 30]      (vị trí 1, 2 — không lấy 3)

nums[:3]     # [10, 20, 30]  (từ đầu đến trước 3)

nums[2:]     # [30, 40, 50]  (từ vị trí 2 đến hết)

nums[::2]    # [10, 30, 50]  (mỗi 2 phần tử một lần)

nums[::-1]   # [50, 40, 30, 20, 10]  (đảo ngược)
```

Content

- Overview
- Foundation
- **Numpy**
- Function

Numpy

- **NumPy** is a library for numerical array computing, and pandas is built upon NumPy.
- For the BA class, only three concepts are required:
  - Array
  - Vectorization—performing calculations on an entire array simultaneously.
  - np.nan

Numpy

Array

NumPy arrays are a part of the NumPy library, which is a tool for numerical computing. Designed for high-performance operations on large datasets and support multi-dimensional arrays and matrices, making them suitable for complex mathematical computations.

- **Multi-dimensional support:** Can handle multiple dimensions, making them suitable for matrix operations and advanced mathematical tasks.
- **Broad broadcasting capabilities:** Allow operations between arrays of different shapes and sizes using broadcasting.
- **Efficient storage and processing:** Uses optimized memory and provides faster performance compared to lists for numerical operations

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

# Element-wise operations
print(a * 2)

# Multi-dimensional array
res = np.array([[1, 2], [3, 4]])
print(res * 2)
```

Output

```
[2 4 6 8]
[[2 4]
 [6 8]]
```

Numpy

Array

- Array indexing and slicing support is identical to that of List.
- The 2D array uses the syntax [**row**, **column**]—essentially a preview of a **data table**.

```
a = np.array([10, 20, 30, 40])

a          # array([10, 20, 30, 40])


a[0]       # 10

a[1:3]     # array([20, 30])
```

```
m = np.array([[1, 2, 3],

                    [4, 5, 6]])

m[0, 1]    # 2   (dòng 0, cột 1)

m[:, 0]    # array([1, 4])  (tất cả dòng, cột 0)
```

Numpy

Vectorization

- Instead of writing loops, NumPy allows for calculations to be performed on the entire array at once.

```
a = np.array([10, 20, 30])

print(a * 2)      # array([20, 40, 60])   — nhân từng phần tử, không cần for

print(a + 5)      # array([15, 25, 35])
```

Numpy

np.nan

- **np.nan** (Not a Number) is the standard way to represent **missing data** in arrays and pandas.

```
b = np.array([1, 2, np.nan, 4])

print(b)        # array([ 1.,  2., nan,  4.])
```

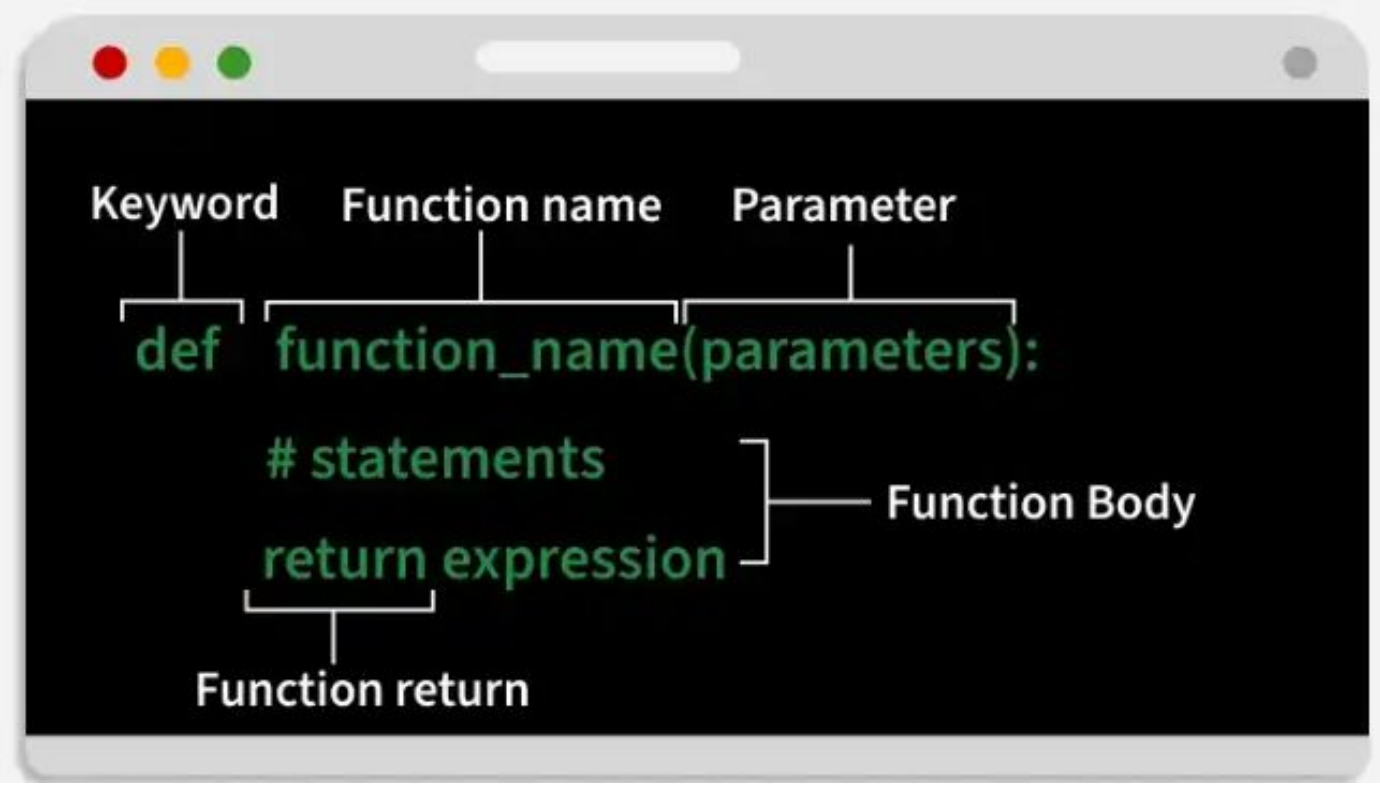
Content

- Overview
- Foundation
- Numpy
- **Function**

Function

Function

- **Python functions** are reusable blocks of code used to perform a specific task.
- They help organize programs into smaller sections and execute the same logic whenever needed by calling the function.
- A function can be defined using **def** keyword. Below is the syntax to define a function



Function

- A function can be defined using **def** keyword. Below is the syntax to define a function

Here, we define a function using def that prints a welcome message when called.

```
def fun():
    print("Welcome to GFG")
```

Calling a Function

After creating a function, call it by using the name of the functions followed by parenthesis containing parameters of that particular function.

```
def fun():
    print("Welcome to GFG")

fun()
```

Output

```
Welcome to GFG
```

Function

Function Arguments

- Arguments are values passed to a function when it is called.
- They allow functions to receive input data and perform operations using those values.

```
def function_name(arguments):
    # function body
    return value
```

- def function_name(arguments): defines a function with optional arguments.
- # function body contains the statements to be executed.
- return value returns a result from the function. If no return statement is used, it returns None by default.

Example: In this example, function checks whether the number passed as an argument is even or odd.

```
def evenOdd(x):
    if (x % 2 == 0):
        return "Even"
    else:
        return "Odd"

print(evenOdd(16))
print(evenOdd(7))
```

Output

```
Even
Odd
```

Thank you!