



Python 3



Agenda

- **Course objective**
- **Python History**
- **Why Python**
- **How Python works**
- **Python syntax**
- **Python Datatypes**
- **Python indentation**
- **Python operators**
- **Falsy values**
- **Control flow**
- **Data structures**
- **LAB1**

Course objective

- Learning Python basic programming syntax in order to use it later in creating different translation layers that can be used in developing and writing different machine learning and AI plugins.



Python history

- Python is a high-level, interpreted programming language created by Guido van Rossum and first released in 1991. The language was inspired by other programming languages like ABC, Modula-3, and UNIX shell scripting languages. Guido van Rossum aimed to create a language that was easy to learn and read, while also powerful enough for complex projects.
- Python was named after the popular British comedy group Monty Python, and the language's humorous and playful nature is reflected in its documentation and community.

- Python has grown in popularity over the years due to its simplicity, readability, and versatility. It has become a popular language for web development, data science, machine learning, scientific computing, and more. The language's popularity has been aided by its large and supportive open-source community, which has created numerous libraries and tools for Python users.
- The latest major version of Python is Python 3, which was released in 2008. Python 2, which was released in 2000, is no longer supported by the language's creators and developers, and users are encouraged to upgrade to Python 3.

Why Python?



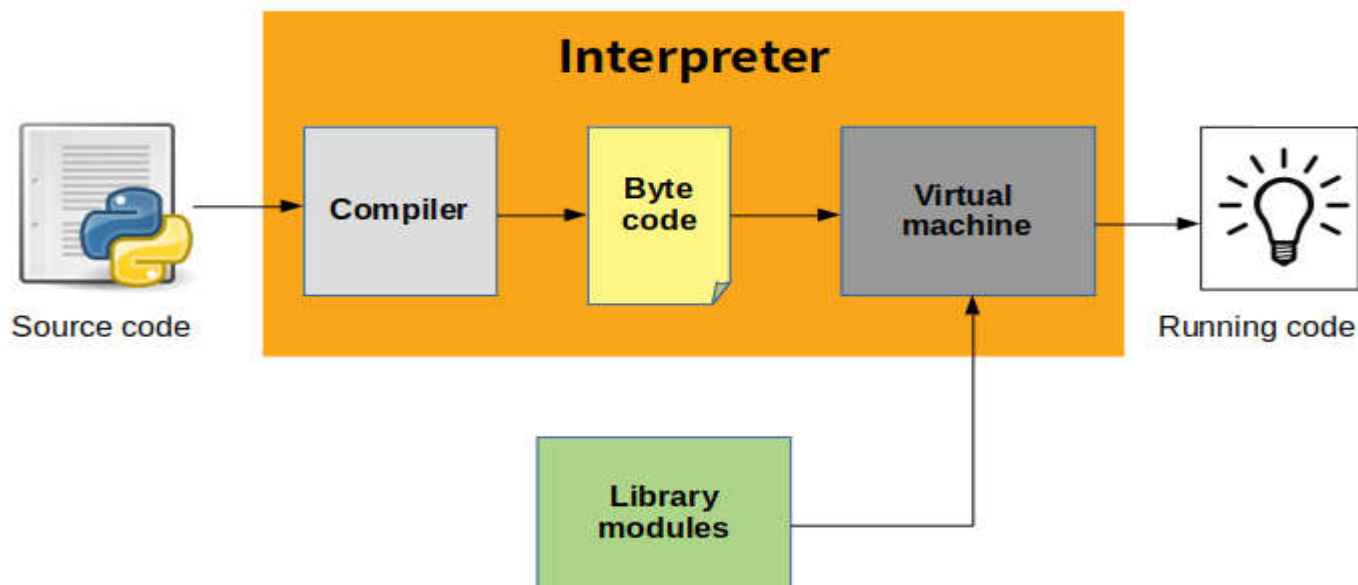
- 1 -Python is a popular programming language for many reasons. Here are important:
- 2 - Easy to Learn: Python is known for its simple and readable syntax, making it easier for beginners to understand and learn.
- 3 - Versatile: Python can be used for a wide range of applications, from web development and data analysis to scientific computing and machine learning.
- 4 - Large and Active Community: Python has a large and supportive open-source community that provides a vast number of libraries and tools that make it easy to get started with Python and to solve problems quickly.
- 5 - Cross-platform Compatibility: Python is available on a variety of platforms, including Windows, Linux, and macOS.
- 6 - Fast Prototyping: Python allows for rapid prototyping, enabling developers to quickly build and test their ideas.

7 - Large Job Market: Python is in high demand in the job market, with many companies and industries seeking Python developers.

Overall, Python is a powerful and versatile language that is easy to learn and has a large community of developers and users. These factors make it a popular choice for a wide range of projects and applications.



How Python works?



1 - Python is a high-level, interpreted programming language that is designed to be easy to read and write. When you write a Python program, the code is first compiled into bytecode, which is a lower-level representation of the program that can be executed by the Python interpreter. Here is a detailed explanation of how Python works:

- Writing Python code: Python code is written in plain text files with the extension ".py". The code is written using the Python syntax, which is designed to be easy to read and write. Python code is organized into modules, which are separate files that can be imported into other Python programs.
- Compiling Python code: When you run a Python program, the source code is first compiled into bytecode. The compilation process is done automatically by the Python interpreter, which converts the source code into bytecode instructions that can be executed by the Python virtual machine.
- Running Python code: Once the code has been compiled into bytecode, the Python interpreter runs the program. The interpreter reads the bytecode instructions one by one and executes them, interpreting the instructions as it goes. The interpreter also

includes a garbage collector, which automatically frees up memory that is no longer being used by the program.

- Python Virtual Machine: The Python interpreter runs on top of the Python virtual machine (VM), which is responsible for executing the bytecode instructions. The VM is designed to be portable and can run on any platform that has a Python interpreter installed. The VM also includes a number of built-in functions and data types that can be used by Python programs.

Overall, Python works by compiling source code into bytecode, which is then interpreted and executed by the Python virtual machine. Its support for multiple programming paradigms, built-in data types and functions, and extensive library support make it a powerful and versatile language for a wide range of applications.

Computers can only understand the code written in machine language. Machine language usually involves instructions written in binary form i.e. 0's & 1's. Hence, in other words, we need a software that can convert our high level code to machine code – we use compiler.

In procedural programming languages like C, compiler directly converts our source code to a machine code. However, in a language like Python, compiler only converts our source code to byte-code. When a python code is compiled, a file (set of instructions) is generated. This is referred to as Byte code. It is Byte code only, which makes python platform independent. Size of each byte code instruction is 1 byte.

Role of Python Virtual Machine (PVM) is to convert the byte code instructions into machine code that are understandable by computers. To do this, PVM has interpreter. Interpreter is responsible for converting byte-code to machine-oriented codes.

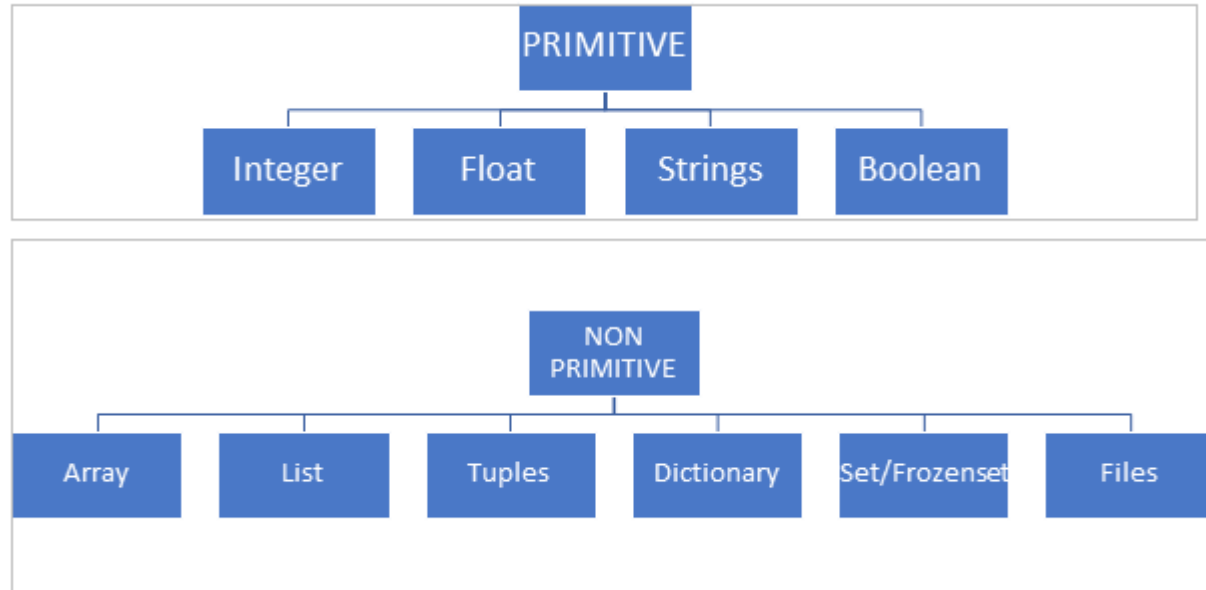


Python syntax

- Python is a programming language that uses a syntax that is relatively to read. Here are some basic syntax rules to keep in mind:
- Statements in Python are typically written on a single line, but you can use a backslash (`\`) to split them into multiple lines.
- Python uses indentation to indicate blocks of code, such as loops or function definitions. Use four spaces to indent each line in a block.
- Python variables can be assigned using the equals sign (`=`). For example, `x = 5` assigns the value 5 to the variable `x`.
- Comments in Python start with a hash symbol (`#`) and continue to the end of the line.

- Python has several types of operators, including arithmetic operators (+, -, *, /), comparison operators (==, !=, <, >, <=, >=), and logical operators (and, or, not).
- Python has several built-in data types, including integers, floating-point numbers, strings, and Booleans.
- Python has several control structures, including if/else statements, loops (for and while), and try/except blocks for error handling.
- Python functions are defined using the "def" keyword, followed by the function name, parameter list (if any), and colon (:). The body of the function is indented.
- Python modules can be imported using the "import" keyword, and specific functions or classes within a module can be imported using the "from" keyword.

Python Datatypes





Python indentation


In Python, indentation refers to the spaces or tabs used at the beginning of a line of code to indicate its level of nesting within a block of code. Python uses indentation to define the scope and structure of its code, instead of using curly braces {} or other characters like many other programming languages.

- Python indentation has some important rules that must be followed:
- Indentation must be consistent throughout the code: All the lines within the same block must be indented by the same number of spaces or tabs.

- The standard indentation is 4 spaces: It is recommended to use 4 spaces for each level of indentation, although tabs can also be used if preferred.
- Indentation is used to create blocks of code: Blocks of code are created by indenting a line after a statement that ends with a colon (:), such as if, for, while, def, class, try, and except. The block continues until the indentation level returns to the previous level.
- Blank lines are ignored: Blank lines at the beginning or end of a block of code are ignored by the interpreter, but blank lines within a block of code must also be indented.

Here is an example of how indentation is used in Python:

python

 Copy code

```
if x > 0:
    print("x is positive")
    if x > 10:
        print("x is greater than 10")
    else:
        print("x is less than or equal to 10")
else:
    print("x is zero or negative")
```

In this example, the if statement creates a block of code, which is indented by 4 spaces. The if statement continues until the indentation level returns to the previous level, which is after the else statement.

Python indentation can sometimes be confusing, especially for beginners. It is important to be consistent with indentation and to pay close attention to the structure of the code. A common mistake is to mix spaces and tabs, which can cause indentation errors. Most modern code editors have an option to automatically convert tabs to spaces or to show invisible characters, which can help avoid such errors.



Python operators:

In Python, operators are symbols or special characters that perform operations on one or more operands, such as variables or values. Python supports a wide range of operators, including arithmetic, comparison, logical, bitwise, and assignment operators.

Here is a brief overview of the different types of Python operators:

- **Arithmetic operators:** These operators perform basic arithmetic operations such as addition, subtraction, multiplication, division, modulus, and exponentiation.
- **Comparison operators:** These operators compare two values and return a Boolean value (True or False) based on the comparison.

- Logical operators: These operators perform logical operations on Boolean values and return a Boolean result. The three logical operators are and, or, and not.
- Bitwise operators: These operators perform bitwise operations on integers and return an integer result. The bitwise operators are AND (&), OR (|), XOR (^), NOT (~), left shift (<<), and right shift (>>).
- Assignment operators: These operators assign a value to a variable and perform an operation on the variable at the same time. For example, += adds a value to a variable and assigns the result to the same variable.

Here is a table of Python operators, grouped by their type:

Operator	Description	Example
Arithmetic Operators		
+	Addition	$3 + 4 = 7$
-	Subtraction	$3 - 4 = -1$
*	Multiplication	$3 * 4 = 12$
/	Division	$3 / 4 = 0.75$
%	Modulus	$3 \% 4 = 3$
**	Exponentiation	$3 ** 4 = 81$
//	Floor division	$3 // 4 = 0$
Comparison Operators		
==	Equal to	$3 == 4 \rightarrow \text{False}$
!=	Not equal to	$3 != 4 \rightarrow \text{True}$
<	Less than	$3 < 4 \rightarrow \text{True}$
>	Greater than	$3 > 4 \rightarrow \text{False}$
<=	Less than or equal to	$3 <= 4 \rightarrow \text{True}$
>=	Greater than or equal to	$3 >= 4 \rightarrow \text{False}$
Logical Operators		
and	Logical AND	$\text{True and False} \rightarrow \text{False}$
or	Logical OR	$\text{True or False} \rightarrow \text{True}$
not	Logical NOT	$\text{not True} \rightarrow \text{False}$
Bitwise Operators		
&	Bitwise AND	$0b1010 \& 0b1100 = 0b1000$
	Bitwise OR	$0b1010 0b1100 = 0b1110$
^	Bitwise XOR	$0b1010 \wedge 0b1100 = 0b0110$
~	Bitwise NOT	$\sim 0b1010 = -0b1011$
<<	Left shift	$0b1010 << 2 = 0b101000$
>>	Right shift	$0b1010 >> 2 = 0b0010$
Assignment Operators		
=	Assign	$x = 3$
+=	Add and assign	$x += 3$ (equivalent to $x = x + 3$)
-=	Subtract and assign	$x -= 3$ (equivalent to $x = x - 3$)
*=	Multiply and assign	$x *= 3$ (equivalent to $x = x * 3$)
/=	Divide and assign	$x /= 3$ (equivalent to $x = x / 3$)
%=	Modulus and assign	



Falsy values


In Python, a value is considered "falsy" if it evaluates to False in a Boolean context. The following values are considered falsy in Python:

- **None:** The None object is a special value in Python that represents the absence of a value. It is considered falsy because it represents no value at all.
- **False:** The Boolean value False is obviously falsy.
- **Zero numerical values:** The integers 0 and 0.0 (zero as a float) are considered falsy.
- **Empty sequences and collections:** Empty strings ("), lists ([]), tuples (()), sets (set()), and dictionaries ({})) are all considered falsy.

- Objects with a `bool()` method that returns `False`: Python objects can define a `bool()` method that returns a Boolean value. If this method returns `False`, the object is considered falsy.
- Objects with a `len()` method that returns `0`: Similarly, objects can define a `len()` method that returns an integer value representing the length of the object. If this method returns `0`, the object is considered falsy.

Here are some examples of falsy values in Python:

python

 Copy code

```
# None
x = None
if not x:
    print("x is falsy")

# False
x = False
if not x:
    print("x is falsy")

# Zero numerical values
x = 0
y = 0.0
if not x and not y:
    print("x and y are falsy")

# Empty sequences and collections
x = ''
y = []
z = ()
w = set()
if not x and not y and not z and not w:
    print("x, y, z, and w are falsy")
```


Control flow:



In Python, control flow refers to the order in which statements are executed in a program. Control flow is governed by three main types of statements: conditional statements, loop statements, and function calls. These statements allow you to control the flow of execution of your program based on certain conditions or user input.


- **Conditional statements:**

Conditional statements allow you to execute certain code blocks only if certain conditions are met. The most commonly used conditional statement in Python is the if statement. The if statement is followed by a condition, which is a Boolean expression that evaluates to True or False.

If the condition is True, the indented code block below the if statement is executed. If the condition is False, the code block is skipped.

Here is an example of an if statement:

python

 Copy code

```
x = 10
if x > 5:
    print("x is greater than 5")
```


In this example, the code block below the if statement is only executed if the condition `x > 5` is True. Since x is equal to 10, which is greater than 5, the code block is executed and the string "x is greater than 5" is printed.

- **Loop statements:**

Loop statements allow you to execute certain code blocks repeatedly, based on certain conditions. The most commonly used loop statement in Python is the for loop. The for loop allows you to iterate over a sequence of values, such as a list or a string. For each value in the sequence, the indented code block below the for loop is executed.

Here is an example of a for loop:

python


 Copy code

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

In this example, the for loop iterates over the list of fruits and prints each one to the console

Another type of loop statement in Python is the while loop. The while loop allows you to execute a code block repeatedly as long as a certain condition is True. Here is an example of a while loop:

python

 Copy code

```
x = 0
while x < 10:
    print(x)
    x += 1
```


In this example, the while loop executes the code block as long as the condition $x < 10$ is True. The code block prints the value of x and then increments it by 1. The loop continues until x is equal to 10, at which point the condition becomes False and the loop terminates.

- **Function calls:**

Function calls allow you to execute a block of code that has been defined elsewhere in your program. Functions are defined using the `def` keyword, followed by the name of the function and a set of parentheses containing any arguments to the function. The code block that makes up the function is indented below the `def` statement.

Here is an example of a function definition:

python

 Copy code

```
def square(x):  
    return x ** 2
```

In this example, the function `square` takes one argument `x` and returns the square of that argument. You can call the function by using its name followed by a set of parentheses containing any arguments to the function. Here is an example of calling the `square` function:

python

 Copy code

```
result = square(3)
print(result)
```

In this example, the `square` function is called with an argument of 3, and the result is assigned to the variable `result`. The value of `result` is then printed to the console, which will output 9.

Overall, control flow is a fundamental concept in programming and is essential to writing effective and efficient programs.

By understanding conditional statements, loop statements, and function calls, you can gain greater control over the execution of your Python code.

Data structures:



Python is a high-level, interpreted programming language that supports many built-in data structures. A data structure is a way of organizing and storing data in a computer's memory. Python data structures are containers that hold data in a specific format, making it easy to access and manipulate the data.


Here are some of the most common data structures in Python:

- **Lists**

Lists are one of the most versatile data structures in Python. They are ordered, mutable, and can hold any type of data, including other lists. Lists are created using square brackets [] and can be accessed using index values.

Example:

python

 Copy code

```
fruits = ['apple', 'banana', 'orange', 'grape']  
print(fruits[1]) # Output: banana
```

Common functions for lists include:


- `append()`: Adds an item to the end of the list
- `insert()`: Adds an item to a specific index
- `remove()`: Removes the first occurrence of an item from the list
- `pop()`: Removes and returns the item at a specific index
- `sort()`: Sorts the list in ascending order

- Tuples

Tuples are similar to lists, but they are immutable, meaning they cannot be changed after they are created. Tuples are created using

Example:

python

 Copy code

```
point = (3, 4)
print(point[0]) # Output: 3
```

parentheses () and can also hold any type of data

Common functions for tuples include:


- `count()`: Returns the number of times a specified value appears in the tuple
- `index()`: Returns the index of the first occurrence of a specified value in the tuple

- **Sets**

Sets are unordered collections of unique elements. Sets are created using curly braces { } or the `set()` function. Sets can be used for mathematical operations like union, intersection, and difference.

Example:

python

 Copy code

```
numbers = {1, 2, 3, 4, 5}  
print(numbers) # Output: {1, 2, 3, 4, 5}
```

Common functions for sets include:

- `add()`: Adds an element to the set
- `remove()`: Removes an element from the set
- `union()`: Returns a set containing all the elements from two or more sets
- `intersection()`: Returns a set containing the elements that are common to two or more sets

- difference(): Returns a set containing the elements that are in one set but not in another

- **Dictionaries**


Dictionaries are unordered collections of key-value pairs. Dictionaries are created using curly braces { } and colons : to separate the keys and values. Keys must be unique and can be of any immutable data type.

Dictionaries

Dictionaries are unordered collections of key-value pairs. Dictionaries are created using curly braces { } and colons : to separate the keys and values. Keys must be unique and can be of any immutable data type.

Example:

python

 Copy code

```
person = {'name': 'Alice', 'age': 30, 'city': 'New York'}  
print(person['name']) # Output: Alice
```

Common functions for dictionaries include:

- `keys()`: Returns a list of all the keys in the dictionary
- `values()`: Returns a list of all the values in the dictionary
- `items()`: Returns a list of all the key-value pairs in the dictionary

These are just a few examples of the many data structures available in Python. By understanding how these data structures work and how to use them effectively, you can write more efficient and powerful Python programs.



LAB1

1. Write a Python function that takes a string as input and returns the string reversed.
2. Write a Python program that prompts the user to enter two numbers and prints out their sum, difference, product, and quotient.
3. Write a Python function that takes a list of integers as input and returns the largest number in the list.
4. Write a Python program that prompts the user to enter their age and prints out a message saying whether they are old enough to vote (age 18 and above) or not.
5. Write a Python function that takes a list of strings as input and returns the string with the most characters.
6. Write a Python program that prompts the user to enter a number and prints out a message saying whether the number is positive, negative, or zero.

7. Write a Python function that takes a string as input and returns the number of vowels in the string.
8. Write a Python program that prompts the user to enter a sentence and prints out the sentence in all uppercase letters.
9. Write a Python function that takes a list of integers as input and returns a new list containing only the even numbers.
10. Write a Python program that reads in a list of strings and prints out the strings in alphabetical order.

Thank you

Hesham Sayed

