Introduction To Python

Dr.Hajialiasgari

Tehran University
Of
Medical Science

October 31, 2024

TEHRAN UNIVERSITY
OF
MEDICAL SCIENCES

Why Python ?

- **Introduction to Python**

Why Python ?

- **Introduction to Python**
- Python is a powerful, easy-to-learn programming language. Known for its clear syntax, its widely used in web development, data analysis, automation, and AI.

Why Python ?

- **Introduction to Python**
- Python is a powerful, easy-to-learn programming language. Known for its clear syntax, its widely used in web development, data analysis, automation, and AI.
- Its versatility and extensive libraries make it ideal for beginners and professionals alike.

Installation

- **Install Python:**
  Download and install Python from:
  `https://www.python.org/downloads/`

Installation

- **Install Python:**
  Download and install Python from:
  `https://www.python.org/downloads/`
- **Install VSCode:**
  Get Visual Studio Code from:
  `https://code.visualstudio.com/`

Installation

- **Install Python:**
  Download and install Python from:
  `https://www.python.org/downloads/`
- **Install VSCode:**
  Get Visual Studio Code from:
  `https://code.visualstudio.com/`
- **Install Jupyter:**
  Use the command:
  `pip install jupyterlab`
  More details at:
  `https://jupyter.org/install`

Constants

- Fixed values such as numbers, letters, and strings, are called constants because their value does not change.

Constants

- Fixed values such as numbers, letters, and strings, are called constants because their value does not change.
- Numeric constants are as you expect.

Constants

- Fixed values such as numbers, letters, and strings, are called constants because their value does not change.
- Numeric constants are as you expect.
- String constants use single quotes (') or double quotes (").

Reserved Words

- False, await, else, import, pass

Reserved Words

- False, await, else, import, pass
- None, break, except, in, raise

Reserved Words

- False, await, else, import, pass
- None, break, except, in, raise
- True, class, finally, is, return

Reserved Words

- False, await, else, import, pass
- None, break, except, in, raise
- True, class, finally, is, return
- and, continue, for, lambda, try

Reserved Words

- False, await, else, import, pass
- None, break, except, in, raise
- True, class, finally, is, return
- and, continue, for, lambda, try
- as, def, from, nonlocal, while

Reserved Words

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

Reserved Words

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

Variables

- A variable is a named place in memory where a programmer can store data and later retrieve it using the variable's name.

Variables

- A variable is a named place in memory where a programmer can store data and later retrieve it using the variable's name.
- Programmers get to choose the names of the variables.

Variables

- A variable is a named place in memory where a programmer can store data and later retrieve it using the variable's name.
- Programmers get to choose the names of the variables.
- You can change the contents of a variable in a later statement.

Python Variable Rules

- Must start with a letter or an underscore (_)

Examples:

- Valid: ‘my_variable‘, ‘var123‘, ‘_myVar‘
- Invalid: ‘123var‘, ‘my-var‘, ‘my var‘

Python Variable Rules

- Must start with a letter or an underscore (_)
- Must consist of letters, numbers, and underscores (_)

Examples:

- Valid: ‘my_variable‘, ‘var123‘, ‘_myVar‘
- Invalid: ‘123var‘, ‘my-var‘, ‘my var‘

Python Variable Rules

- Must start with a letter or an underscore (_)
- Must consist of letters, numbers, and underscores (_)
- Case sensitive (e.g., ‘var‘ and ‘Var‘ are different variables)

Examples:

- Valid: ‘my_variable‘, ‘var123‘, ‘_myVar‘
- Invalid: ‘123var‘, ‘my-var‘, ‘my var‘

Different Types in Python

- Python supports different data types, including:

Examples:

- String: ‘"Hello World"‘
- Integer: ‘42‘
- Float: ‘3.14159‘
- Complex: ‘1+2j‘

Syntax:

- type (42)
  <class int>

Different Types in Python

- Python supports different data types, including:
  - **String (str)**: A sequence of characters, e.g., ‘’Hello’‘, ‘"World"‘

Examples:

- String: ‘"Hello World"‘
- Integer: ‘42‘
- Float: ‘3.14159‘
- Complex: ‘1+2j‘

Syntax:

- type (42)
  <class int>

Different Types in Python

- Python supports different data types, including:
  - **String (str)**: A sequence of characters, e.g., ‘’Hello’‘, ‘"World"‘
  - **Integer (int)**: Whole numbers, e.g., ‘10‘, ‘-5‘

Examples:

- String: ‘"Hello World"‘
- Integer: ‘42‘
- Float: ‘3.14159‘
- Complex: ‘1+2j‘

Syntax:

- type (42)
  <class int>

Different Types in Python

- Python supports different data types, including:
  - **String (str)**: A sequence of characters, e.g., ‘’Hello’‘, ‘"World"‘
  - **Integer (int)**: Whole numbers, e.g., ‘10‘, ‘-5‘
  - **Float**: Numbers with decimal points, e.g., ‘3.14‘, ‘-0.001‘

Examples:

- String: ‘"Hello World"‘
- Integer: ‘42‘
- Float: ‘3.14159‘
- Complex: ‘1+2j‘

Syntax:

- type (42)
  <class int>

Different Types in Python

- Python supports different data types, including:
  - **String (str)**: A sequence of characters, e.g., ‘’Hello’‘, ‘"World"‘
  - **Integer (int)**: Whole numbers, e.g., ‘10‘, ‘-5‘
  - **Float**: Numbers with decimal points, e.g., ‘3.14‘, ‘-0.001‘
  - **Complex**: Complex numbers, e.g., ‘1+2j‘

Examples:

- String: ‘"Hello World"‘
- Integer: ‘42‘
- Float: ‘3.14159‘
- Complex: ‘1+2j‘

Syntax:

- type (42)
  <class int>

Different Types in Python

- Python supports different data types, including:
  - **String (str)**: A sequence of characters, e.g., ‘’Hello’‘, ‘"World"‘
  - **Integer (int)**: Whole numbers, e.g., ‘10‘, ‘-5‘
  - **Float**: Numbers with decimal points, e.g., ‘3.14‘, ‘-0.001‘
  - **Complex**: Complex numbers, e.g., ‘1+2j‘
  - **Boolean**: True , False

Examples:

- String: ‘"Hello World"‘
- Integer: ‘42‘
- Float: ‘3.14159‘
- Complex: ‘1+2j‘

Syntax:

- type (42)
  <class int>

Operators and Operations

Operator	Operation
+	Addition
-	Subtraction
*	Multiplication
/	Division
**	Exponentiation
//	Floor Division
%	Modulus (Remainder)

Examples of Operations

Addition and Subtraction:

- `5 + 3 = 8`

Examples of Operations

Addition and Subtraction:

- `5 + 3 = 8`
- `10 - 4 = 6`

Examples of Operations

Multiplication and Division:

- `4 * 3 = 12`

Examples of Operations

Multiplication and Division:

- `4 * 3 = 12`
- `8 / 2 = 4.0`

Examples of Operations

Exponentiation, Floor Division, Modulus:

- `2 ** 3 = 8`

Examples of Operations

Exponentiation, Floor Division, Modulus:

- `2 ** 3 = 8`
- `7 // 2 = 3`

Examples of Operations

Exponentiation, Floor Division, Modulus:

- `2 ** 3 = 8`
- `7 // 2 = 3`
- `7 % 2 = 1`

Order of Evaluation

- When we string operators together, Python must know which one to do first.

Order of Evaluation

- When we string operators together, Python must know which one to do first.
- This is called operator precedence.

Order of Evaluation

- When we string operators together, Python must know which one to do first.
- This is called operator precedence.
- Which operator takes precedence over the others?

Operator Precedence Rule

Highest precedence rule to lowest precedence rule:

- Parentheses are always respected: `(2 + 3) * 4 = 20`

Operator Precedence Rule

Highest precedence rule to lowest precedence rule:

- Parentheses are always respected: `(2 + 3) * 4 = 20`
- Exponentiation (raise to a power): `2 ** 3 = 8`

Operator Precedence Rule

Highest precedence rule to lowest precedence rule:

- Parentheses are always respected: `(2 + 3) * 4 = 20`
- Exponentiation (raise to a power): `2 ** 3 = 8`
- Multiplication, Division, and Remainder: `8 / 2 * 4 = 16`

Operator Precedence Rule

Highest precedence rule to lowest precedence rule:

- Parentheses are always respected: `(2 + 3) * 4 = 20`
- Exponentiation (raise to a power): `2 ** 3 = 8`
- Multiplication, Division, and Remainder: `8 / 2 * 4 = 16`
- Addition and Subtraction: `3 + 5 - 2 = 6`

Operator Precedence Rule

Highest precedence rule to lowest precedence rule:

- Parentheses are always respected: `(2 + 3) * 4 = 20`
- Exponentiation (raise to a power): `2 ** 3 = 8`
- Multiplication, Division, and Remainder: `8 / 2 * 4 = 16`
- Addition and Subtraction: `3 + 5 - 2 = 6`
- Left to right: `2 + 3 * 4 = 14`

User Input

- We can instruct Python to pause and read data from the user using the `input()` function.

Example:

```
name = input("Enter your name: ")
print("Hello, " + name)
```

User Input

- We can instruct Python to pause and read data from the user using the `input()` function.
- The `input()` function returns a string.

Example:

```
name = input("Enter your name: ")
print("Hello, " + name)
```

Converting Input

- If we want to read a number from the user, we must convert it from a string to a number using a type conversion function.

Example:

```
age = input("Enter your age: ")
age = int(age)
print(age + 5)
```

Converting Input

- If we want to read a number from the user, we must convert it from a string to a number using a type conversion function.
- Later we will deal with bad input data.

Example:

```
age = input("Enter your age: ")
age = int(age)
print(age + 5)
```

Conditional Execution

- Conditional execution allows the program to execute certain blocks of code based on specific conditions.

Conditional Execution

- Conditional execution allows the program to execute certain blocks of code based on specific conditions.
- The primary conditional statements in Python are `if`, `elif`, and `else`.

Comparison Operators

Operator	Description	Example
==	Equal to	`a == b`
!=	Not equal to	`a != b`
>	Greater than	`a > b`
<	Less than	`a < b`
>=	Greater than or equal to	`a >= b`
<=	Less than or equal to	`a <= b`

If Statement

- The `if` statement checks a condition and executes the block of code if the condition is true.

Example:

```
if age >= 18:
    print("You are an adult")
```

Else Statement

- The `else` statement executes if all preceding conditions are false.

Example:

```
if temperature > 30:
    print("It's a hot day.")
else:
    print("It's a pleasant day.")
```

Nested Conditionals

- You can nest `if` statements within other `if` statements to create more complex conditions.

Example:

```
if score >= 50:
    print("You passed.")
    if score >= 90:
        print("Excellent!")
    else:
        print("Good job.")
else:
    print("You failed.")
```

End of session 1