Python - Quick Guide for Ultimate Python Beginner's

Connect with me: Youtube | LinkedIn | WhatsApp Channel | Web | Facebook | Twitter

- Download PDF
- To access the updated handouts, please click on the following link: https://yasirbhutta.github.io/ms-excel/docs/quick-guide.html
- Pyton Resources: Books, Websites, Tutorials

What is Python

- Python is a high-level, general-purpose programming language.
- It is known for its clear syntax, readability, and versatility.
- Python is widely used for `web development`, `data science`, `machine learning`, and `automation`.

Getting Started

- Install Python: Download and install it from https://www.python.org/downloads/.
- Choose a text editor: A program to write code, like Visual Studio Code, Jupyter Notebook, `PyCharm`, or even a simple text editor like `Notepad`.
- Text editor for Android: Pydroid 3 - IDE for Python 3
  - Video: How to: Install Jupyter Notebook on an Android device
- Interactive mode: Experiment with Python directly in your terminal or command prompt using the python command.

Basics

Python source code

> **Important:** Python source code files always use the `.py` extension.

How To Use Print() Function in Python

- It is used to display text to the console, or to a file. The print() function can take one or more arguments, and it can be used to format text in a variety of ways.

Example #1:

```
message = 'Python is fun'

# print the string message
print(message)
```

Output:

```
Python is fun
```

Example #2:

```
# Print a string:
print("Hello, World!")

# Print a number:
print(10)

# Print a variable:
x = 5
print(x)

# Print multiple objects on the same line:
print("Hello", "World")

# Print multiple objects on separate lines:
print("Hello")
print("World")

# Print with a custom separator:
print("Hello", "World", sep=", ")

# Print with a custom ending character:
print("Hello", "World", end="!")
```

See also:

- Video: How to print multiple lines
- Video: 100 times "hello world" without loop

Variables

- Storage containers for data (numbers, text, etc.).

What is a variable

- A variable is a named storage location in a computer's memory that is used to hold data or values. It allows programmers to store and manipulate data within a program.

Purpose: Variables provide a way to store and manage data that can be used and manipulated throughout a program. They make programs more flexible and allow for dynamic data storage.

Assignment statement: in Python is used to assign a value to a variable. Its primary purpose is to store and manage data within a program.

Imagine variables as labeled boxes:

- You have boxes for storing different things (numbers, words, etc.).
- Each box has a name (label) to identify what's inside.
- You can put things in, take them out, and change what's inside.

Example #3: Storing a name

```
name = "Muhammad Hamza"
print(name)
```

Example #4: Tracking a score:

```
score = 0
score = score + 10 # adds 10 to the score
print(score)
```

Example #5: Remembering a favorite color

```
favorite_color = "blue" #stores "blue"  in variable
print(favorite_color)
```

Example #6: Calculating the area of a rectangle

```
length = 10
width = 5

# calculates the area
area = length * width
print(area)
```

Key Points:

- **Choose meaningful names:** Use names that describe what the variable stores (e.g., pizza_slices instead of x).
- **Assign values using =:** The equals sign is used to put a value into a variable.
- **Change values:** You can update a variable's value later in your code.
- **Use variables in calculations and operations:** Variables can be used just like regular numbers or text in expressions.
- **Think of variables as placeholders:** They hold information that can change as your program runs.

See also:

- Variables in Python

Types of Data

- Numbers (integers, floats), strings (text), booleans (True/False)

String (str)

- Stores text or characters.
- Enclosed in single or double quotes.

Example #7:

```
name = "Ahmad"
greeting = 'Hello, world!'
favorite_song = "Let It Go"  # Double quotes can also contain single quotes
```

Integer (int)

- Stores whole numbers (without decimals).
- Used for counting and representing quantities.

Example #8:

```
age = 25
num_pets = 3
lucky_number = 7
```

Float (float)

- Stores decimal numbers.
- Used for representing precise measurements or values with fractions

Example #9:

```
height = 1.70  # meters
price = 19.99
distance = 3.14159  # Pi
```

Boolean (bool)

- Stores True or False values.
- Used for representing logical conditions or decisions.

Example #9:

```
is_hungry = True
is_raining = False
```

```
has_finished = True
```

Key Points

- Each data type has specific purposes and operations.
- Python automatically determines the data type when you assign a value.
- You can check the data type using the type() function:

```
print(type(name))  # Output: <class 'str'>
print(type(age))  # Output: <class 'int'>
print(type(height))  # Output: <class 'float'>
print(type(is_hungry))  # Output: <class 'bool'>
```

See also:

- Video: Use of type() function in Python

Operators

- Perform calculations and comparisons (e.g., +, -, *, /, ==, !=).

1. Arithmetic Operators:

- Used for performing basic mathematical calculations.
- Operators: `+` (addition), `-` (subtraction), `*` (multiplication), `/` (division), `//` (floor division), `%` (modulo), `**` (exponentiation)

Example:

```
result = 10 + 5  # Addition
difference = 15 - 7  # Subtraction
product = 4 * 6  # Multiplication
quotient = 12 / 3  # Division
integer_quotient = 17 // 4  # Floor division
remainder = 25 % 4  # Modulo
square = 5 ** 2  # Exponentiation
```

2. Comparison Operators:

- Used to compare values and return a Boolean result (True or False).

- Operators: `==` (equal to), `!=` (not equal to), `>` (greater than), `<` (less than), `>=` (greater than or equal to), `<=` (less than or equal to)

- Examples:

```
is_equal = 7 == 7  # True
is_greater = 12 > 9  # True
is_less_or_equal = 5 <= 5  # True
```

Input and Output

- Get user input with input() and display output with print().

While I can't create physical handouts, I can provide you with a comprehensive guide to Python input and output, suitable for learning and reference:

Input in Python:

- **The `input()` function:**
  - Takes a prompt as an argument (optional)
  - Reads user input from the keyboard
  - Returns the input as a string
- **Example:**

```
name = input("Enter your name: ")
age = int(input("Enter your age: "))  # Convert string input to integer
```

Output in Python:

- **The `print()` function:**
  - Prints values to the console
  - Can display multiple values, separated by commas
  - Can format output using string methods
- **Examples:**

```
print("Hello, world!")
print("Your name is", name, "and you are", age, "years old.")
print("The answer is:", 42)
```

Control Flow

Conditional Statements

if statement

- The if statement in MATLAB is a conditional statement that allows you to execute a block of code only if a certain condition is met.

- Make decisions using `if`, `elif`, and `else` statements.

The general **syntax** of the if statement is as follows:

```
if condition
    statements
```

The `condition` can be any logical expression. If the condition is evaluated to `true`, the block of `statements` is executed. Otherwise, the block of `statements` is skipped.

Here is a simple example of an if statement in MATLAB:

```
x = 10

if x > 5:
    print('x is greater than 5.')
```

This code will print the message `x is greater than 5.` to the console.

You can also use `elif` statements to check for multiple conditions. The general **syntax** of the **elif statement** is as follows:

```
elif condition
    statements
end
```

If the `condition` for the if statement is evaluated to `false`, the python interpreter will check the `condition` for the first elif statement. If the condition for the elif statement is evaluated to `true`, the corresponding block of `statements` is executed. Otherwise, the python interpreter will check the `condition` for the next elif statement, and so on.

Here is an example of an if statement with an elif statement:

```
x = 3

if x > 5:
    print('x is greater than 5.')
elif x < 5:
    print('x is less than 5.')
```

This code will print the message "x is less than 5." to the console.

You can also use an else statement to check for all other conditions. The general syntax of the else statement is as follows:

```
else
    statements
end
```

If all of the conditions for the if and elseif statements are evaluated to `false`, the block of `statements` in the `else` statement is executed.

Here is an example of an if statement with an `elif` statement and an `else` statement:

```
x = 2

if x > 5:
    print('x is greater than 5.')
elif x == 5:
    print('x is equal to 5.')
else:
    print('x is less than 5.')
```

This code will print the message "x is less than 5." to the console.

Loops

- Repeat actions using `for` and `while` loops.

for loop

- A for loop in Python is a programming statement that repeats a block of code a certain number of times.

Example:

```
for i in range(5):
    print(i)
```

Example:

```
for i in range(5):
    print("Python")
```

while loop

- A while loop in python is a control flow statement that repeatedly executes a block of code until a specified condition is met.

Example: Counting Up to a Number:

```
count = 1  # Start counting at 1
while count <= 10:  # Keep counting as long as we're less than or equal to 10
    print(count)  # Print the current number
    count += 1  # Add 1 to the count for the next round
```

True/False (Mark T for True and F for False)

Multiple Choice (Select the best answer)

Exercises

Review Questions

References and Bibliography