# Python Programming

asaad.nayyef@uomus.edu.iq

## Week 1: Introduction to Python, Variables, Data Types, and Basic Operators

### Basic input

A **computer** is an electronic device that stores and processes information. Examples of computers include smartphones, tablets, laptops, desktops, and servers. Technically, a **program** is a sequence of instructions that a computer can run. Programs help people accomplish everyday tasks, create new technology, and have fun.

**The Python language**

one of the top programming languages today. Leading tech giants like Google, Apple, NASA, Instagram, Pixar, and others use Python extensively.

One reason why Python is popular is because many libraries exist for doing real work. A **library** is a collection of code that can be used in other programs. Python comes with an extensive Standard Library for solving everyday computing problems like extracting data from files and creating summary reports. In addition, the community develops many other libraries for Python.

Another reason why Python is popular is because the syntax is concise and straightforward. The **syntax** of a language defines how code must be structured. Syntax rules define the keywords, symbols, and formatting used in programs. Compared to other programming languages, Python is more concise and straightforward.

**Hello world in Python and Java**

By tradition, Hello World is the first program to write when learning a new language. This program simply displays the message "Hello, World!" to the user. The hello world
program is only one line in Python:

```
print("Hello, World!")
```

In contrast, the hello world program is five lines in Java (a different language).

```
public class Hello {
public static void main(String[] args) {
System.out.println("Hello, World!");
}
}
```

# Input/output

## Learning objectives

By the end of week 1 you should be able to

- Display output using the print() function.
- Obtain user input using the input() function.

## Basic output

The **print()** function displays output to the user. **Output** is the information or result produced by a program. The sep and end options can be used to customize the output. Table 1.1 shows examples of sep and end.

Multiple values, separated by commas, can be printed in the same statement. By default, each value is separated by a space character in the output. The sep option can be used to change this behavior.

By default, the print() function adds a newline character at the end of the output. A **newline** character tells the display to move to the next line. The end option can be used to continue printing on the same line.

| Code | Output |
|---|---|
| `print("Today is Monday.")`<br>`print("I like string beans.")` | `Today is Monday.`<br>`I like string beans.` |
| `print("Today", "is", "Monday")`<br>`print("Today", "is", "Monday", sep="...")` | `Today is Monday`<br>`Today...is...Monday` |
| `print("Today is Monday, ", end="")`<br>`print("I like string beans.")` | `Today is Monday, I like string beans.` |
| `print("Today", "is", "Monday", sep="? ", end="!!")`<br>`print("I like string beans.")` | `Today? is? Monday!!I like string beans.` |

## Basic input

Computer programs often receive input from the user. **Input** is what a user enters into a program. An input statement, variable = input("prompt"), has three parts:

1. A **variable** refers to a value stored in memory. In the statement above, variable can be replaced with any name the programmer chooses.
2. The **input()** function reads one line of input from the user. A function is a named, reusable block of code that performs a task when called. The input is stored in the computer's memory and can be accessed later using the variable.
3. A **prompt** is a short message that indicates the program is waiting for input. In the statement above, "prompt" can be omitted or replaced with any message.

Ex1

```
variable = input("prompt")
```

**Program Output**

Prompt

Ex2

```
print("Please enter a number: ")
number = input()
print("Value =", number)
```

**Program Output**

Please enter a number:
1
Value = 1

# Variables

## Learning objectives

By the end of this week 1 you should be able to
• Assign variables and print variables.
• Explain rules for naming variables.

## Assignment statement

Variables allow programs to refer to values using names rather than memory locations. Ex: age refers to a person's age, and birth refers to a person's date of birth.
A statement can set a variable to a value using the **assignment operator** (=). Note that this is different from the equal sign of mathematics. Ex: age = 6 or birth = "May 15". The left side of the assignment statement is a variable, and the right side is the value the variable is assigned.
Ex3

```
city = "Baghdad"
print("The city where you live is", city)
```

**Program Output**

The city where you live is Baghdad

## Variable naming rules

A variable name can consist of letters, digits, and underscores and be of any length. The name cannot start with a digit. Ex: 101class is invalid. Also, letter case matters. Ex: Total is different from total. Python's style guide recommends writing variable names in **snake case**, which is all lowercase with underscores in between each word, such as first_name or total_price.
A name should be short and descriptive, so words are preferred over single characters in programs for readability. Ex: A variable named count indicates the variable's purpose better than a variable named c. Python has reserved words, known as **keywords**, which have special functions and cannot be used as names for variables (or other objects).

**Table** The Python key words

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

**Table Keywords**

## Numeric data types

Python supports two basic number formats, **integer and floating-point**. An integer represents a whole number, and a floating-point format represents a decimal number. The format a language uses to represent data is called a data type. In addition to integer and floating-point types, programming languages typically have a string type for representing text.

## Assigning values to variables

**Table** Python math operators

| Symbol | Operation | Description |
|---|---|---|
| + | Addition | Adds two numbers |
| - | Subtraction | Subtracts one number from another |
| * | Multiplication | Multiplies one number by another |
| / | Division | Divides one number by another and gives the result as a floating-point number |
| // | Integer division | Divides one number by another and gives the result as an integer |
| % | Remainder | Divides one number by another and gives the remainder |
| ** | Exponent | Raises a number to a power |

## Operator precedence

When a calculation has multiple operators, each operator is evaluated in order of precedence. Ex: 1 + 2 * 3 is 7 because multiplication takes precedence over addition. However, (1 + 2) * 3 is 9 because parentheses take precedence over multiplication.

| Operator | Description | Example | Result |
|---|---|---|---|
| () | Parentheses | (1 + 2) * 3 | 9 |
| ** | Exponentiation | 2 ** 4 | 16 |
| +, - | Positive, negative | -math.pi | -3.14159 |
| *, / | Multiplication, division | 2 * 3 | 6 |
| +, - | Addition, subtraction | 1 + 2 | 3 |

**Table Operator precedence from highest to lowest.**

## Relational operators

**Table** Relational operators

| Operator | Meaning |
|---|---|
| > | Greater than |
| < | Less than |
| >= | Greater than or equal to |
| <= | Less than or equal to |
| == | Equal to |
| != | Not equal to |

**Table** Boolean expressions using relational operators

| Expression | Meaning |
|---|---|
| `x > y` | Is x greater than y? |
| `x < y` | Is x less than y? |
| `x >= y` | Is x greater than or equal to y? |
| `x <= y` | Is x less than or equal to y? |
| `x == y` | Is x equal to y? |
| `x != y` | Is x not equal to y? |

**The 6 comparison operators:**

- **equal to: ==**
- **not equal to: !=**
- **greater than: >**
- **less than: <**
- **greater than or equal to: >=**
- **less than or equal to: <=**

**Table** Algebraic and programming expressions

| Algebraic Expression | Python Statement |
|---|---|
| $y = 3\frac{x}{2}$ | `y = 3 * x / 2` |
| $z = 3bc + 4$ | `z = 3 * b * c + 4` |
| $a = \frac{x + 2}{b - 1}$ | `a = (x + 2) / (b - 1)` |

**Table** Some expressions

| Expression | Value |
|---|---|
| `5 + 2 * 4` | 13 |
| `10 / 2 – 3` | 2 |
| `8 + 12 * 2 – 4` | 28 |
| `6 – 3 * 2 + 7 – 1` | 6 |

**Table** More expressions and their values

| Expression | Value |
|---|---|
| `(5 + 2) * 4` | 28 |
| `10 / (5 – 3)` | 5 |
| `8 + 12 * (6 – 2)` | 56 |
| `(6 – 3) * (2 + 7) / 3` | 9 |

## 1. Simple arithmetic operations:

**Four basic arithmetic operators exist in Python:**

**1. Addition (+)**
**2. Subtraction (-)**
**3. Multiplication (*)**
**4. Division (/)**

```
a = 10  # Integer variable
b = 20  # Integer variable # Variables a and b are assigned integer values.
# Performing arithmetic operations
print("Sum:", a + b)  # Adds a and b function displays the sum of a and b.

print("Difference:", a - b) # Subtracts b from a function displays the difference between a and b
print("Division:", b / a)  # Division b and a
print("Integer division:", b // a)  # Integer division b and a
print("Remainder:", b % a)  # Remainder a % b
print("Raises a number:", 3 ** 2)
```

**Program Output**

```
Sum: 30
Difference: -10
Division: 2.0
Integer division: 2
Remainder: 0
Raises a number: 9
```

## 2. Comparison operators:

```
x = 5
```

**y = 10 #ariables x and y are assigned integer values.**
**print(x > y)  # Checks if x is greater than y**
**print(x <= y)  # Checks if x is less than or equal to y**
**Run**
False
True

## 3. Logical operators:

**TABLE** Truth Table for Basic Logical Operators

| p | q | p == q | p != q | p and q | p or q | not p |
|---|---|---|---|---|---|---|
| False | False | True | False | False | False | True |
| False | True | False | True | False | True | True |
| True | False | False | True | False | True | False |
| True | True | True | False | True | True | False |

Ex
is_sunny = True
is_warm = False
print(is_sunny and is_warm)  # Checks if both conditions are True
print(is_sunny or is_warm)  # Checks if at least one condition is True
**Program Output**
False
True

**Table** Some of Python's escape characters

| Escape Character | Effect |
|---|---|
| \n | Causes output to be advanced to the next line. |
| \t | Causes output to skip over to the next horizontal tab position. |
| \' | Causes a single quote mark to be printed. |
| \" | Causes a double quote mark to be printed. |
| \\ | Causes a backslash character to be printed. |

**The \t escape character advances the output to the next horizontal tab position. (A tab position normally appears after every eighth character.) The following statements are illustrative:**

```
print('Mon\tTues\tWed')
print('Thur\tFri\tSat')
```

**This statement prints Monday, then advances the output to the next tab position, then prints Tuesday, then advances the output to the next tab position, then prints Wednesday. The output will look like this:**

**Mon  Tues  Wed**
**Thur  Fri  Sat**

## How to read errors

A natural part of programming is making mistakes. Even experienced programmers make mistakes when writing code. Errors may result when mistakes are made when writing code. The computer requires very specific instructions telling the computer what to do. If the instructions are not clear, then the computer does not know what to do and gives back an error.
When an error occurs, Python displays a message with the following information:
1. The line number of the error.
2. The type of error (Ex: SyntaxError).
3. Additional details about the error.
Ex: Typing print "Hello!" without parentheses is a syntax error. In Python, parentheses are required to use print. When attempting to run print "Hello!", Python displays the following error:

```
print "Hello!"
```

```
Traceback (most recent call last):
File "C:\PycharmProjects\PythonProjectErrors\main.py", line 1
    print "Hello!"
    ^^^^^^^^^^^^^^
SyntaxError: Missing parentheses in call to 'print'. Did you mean print(...)?
```

**Common types of errors**

Different types of errors may occur when running Python programs. When an error occurs, knowing the type of error gives insight about how to correct the error. The following table shows examples of mistakes that anyone could make when programming.

| Mistake | Error message | Explanation |
|---|---|---|
| `print("Have a nice day!"` | `SyntaxError: unexpected EOF while parsing` | The closing parenthesis is missing. Python is surprised to reach the end of file (EOF) before this line is complete. |
| `word = input("Type a word: )` | `SyntaxError: EOL while scanning string literal` | The closing quote marks are missing. As a result, the string does not terminate before the end of line (EOL). |
| `print("You typed:", wird)` | `NameError: name 'wird' is not defined` | The spelling of `word` is incorrect. The programmer accidentally typed the wrong key. |

| | | |
|---|---|---|
| `prints("You typed:", word)` | `NameError: name 'prints' is not defined` | The spelling of `print` is incorrect. The programmer accidentally typed an extra letter. |
| `  print("Hello")` | `IndentationError: unexpected indent` | The programmer accidentally typed a space at the start of the line. |
| `    print("Goodbye")` | `IndentationError: unexpected indent` | The programmer accidentally pressed the Tab key at the start of the line. |

**Table Simple mistakes.**

```
print("Have a nice day!"
```

```
File "G:\PycharmProjects\PythonProjectErrors\main.py", line 3
    print("Have a nice day!"
         ^
SyntaxError: '(' was never closed
```

```
word = input("Type a word: )
```

```
File "C:\PycharmProjects\PythonProjectErrors\main.py", line 5
    word = input("Type a word: )
                 ^
SyntaxError: unterminated string literal (detected at line 5)
```

```
word = input("Type a word:" )
print("You typed:", wird)
```

```
Traceback (most recent call last):
  File "G:\PycharmProjects\PythonProjectErrors\main.py", line 6, in <module>
    print("You typed:", wird)
                        ^^^^
NameError: name 'wird' is not defined. Did you mean: 'word'?
```

```
word = input("Type a word:" )
prints("You typed:", word)
```

```
Traceback (most recent call last):
  File "C:\PycharmProjects\PythonProjectErrors\main.py", line 8, in <module>
    prints("You typed:", word)
    ^^^^^^
NameError: name 'prints' is not defined. Did you mean: 'print'?
```

```
  print("Hello")
```

```
File "C:\PycharmProjects\PythonProjectErrors\main.py", line 10
    print("Hello")
IndentationError: unexpected indent
```

```
    print("Hello")
```

File "C:\PycharmProjects\PythonProjectErrors\main.py", line 10
print("Hello")
IndentationError: unexpected indent

**The hash character**

**Comments** are short phrases that explain what the code is doing. In the following program contain comments. Each comment begins with a hash character (#). All text from the hash character to the end of the line is ignored when running the program. In contrast, hash characters inside of strings are treated as regular text.

```
#print "Hello!"

#print("Have a nice day!"

#word = input("Type a word:" )
#print("You typed:", word)
#ord = input("Type a word:" )
#prints("You typed:", word)

print("Welcome to Al Mutagbal University")
```

C:\PycharmProjects\PythonProjectErrors\main.py

Welcome to Al Mutagbal University

## Floating-point errors

Computers store information using 0's and 1's. All information must be converted to a string of 0's and 1's. Ex: 5 is converted to 101. Since only two values, 0 or 1, are allowed the format is called binary. Floating-point values are stored as binary by Python. The conversion of a floating point number to the underlying binary results in specific types of floating-point errors.
A **round-off error** occurs when floating-point values are stored erroneously as an approximation. The difference between an approximation of a value used in computation and the correct (true) value is called a round-off error