

IT111 Introduction to Information Systems

Eng. Eman Hamdy

جامعة الجميع الذكية

تسعى لتكون الكيان الأول الذي يتبنى مبدأ المجتمع التعليمي
التعاوني ويقدم العلم مقابل العمل للتمهيد وبناء المجتمعات

Python Programming Language



جامعة الجميع الذكية
EVERYONE'S SMART UNIVERSITY

What is Programming?

1. Just like we humans can understand a few languages (Arabic, English, etc.), so is the case with computers.
2. Computers understand instructions that are written in a specific syntactical form called a programming language.
3. Simply, programming is a way to tell computers to do a specific task.
4. Some of the popular Programming languages are Python, C, C++, Java, etc

What is Python?

Python is a popular programming language. It was created by Guido van Rossum, and released in 1991.

- It is used for:
 1. web development (server-side),
 2. software development,
 3. system scripting.

Why Python?

1. Python works on different platforms (Windows, Mac, Linux, Raspberry Pi, etc).
2. Python has a simple syntax similar to the English language.
3. Python has syntax that allows developers to write programs with fewer lines than some other programming languages.

1. Python was designed for readability and has some similarities to the English language with influence from mathematics.
2. Python uses new lines to complete a command, as opposed to other programming languages which often use semicolons or parentheses.
3. Python relies on indentation, using whitespace, to define scope; such as the scope of loops, functions and classes. Other programming languages often use curly-brackets for this purpose.

High-Level Languages Python Success

- From IEEE spectrum **2017**: programming languages ranking

Language Rank	Types	Spectrum Ranking
1. Python	 	100.0
2. C	  	99.7
3. Java	  	99.5
4. C++	  	97.1
5. C#	  	87.7
6. R		87.7
7. JavaScript	 	85.6
8. PHP		81.2
9. Go	 	75.1
10. Swift	 	73.7

Variables

- In a Python program, **variables** are used to **store values**.
- A **variable** is a **storage location in a computer program**. Each variable has a **name** and holds a **value**
- A variable is similar to a parking space in a parking garage. The parking space has an identifier (as “J 053”), and it can hold a vehicle.



Javier Larrea/Age Fotostock.

Like a variable in a computer program, a parking space has an identifier and a contents.



Variable Names

- A variable is defined when you give it a name that explains its purpose.
- Variable Name must follow a few simple rules:
 - Names must start with a letter or the underscore (_) character, and the remaining characters must be letters, numbers, or underscores
 - Cannot use other symbols such as ? or %.
 - Spaces are not permitted inside names either.
- Names are **case sensitive**, that is, **canVolume** and **canvolume** are different names.
- You cannot use **reserved words** such as **if** or **class** as names; these words are reserved exclusively for their special Python meanings

- The left-hand side of an assignment statement consists of a variable.
- The right-hand side is an expression that has a value. That value is stored in the variable.
- Example: ***cansPerPack = 6*** here a variable name *canPerPack* will be equal to 6 after running this assignment statement
- **The first time a variable is assigned a value, the variable is created and initialized with that value.**

- After a variable has been defined, it can be used in other statements. E.g., **print(cansPerPack)**
- It will print the value stored in the variable cansPerPack
- If an existing variable is **assigned a new value**, that value **replaces** the previous contents of the variable. Example:

cansPerPack = 8

1 Because this is the first assignment, the variable is created.

cansPerPack =

2 The variable is initialized.

cansPerPack =

3 The second assignment overwrites the stored value.

cansPerPack =

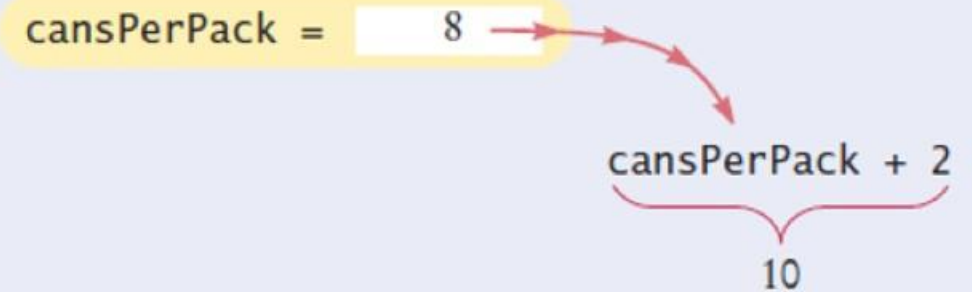
Figure 1 Executing Two Assignments

- Consider this example:

cansPerPack = cansPerPack + 2

- This statement means to look up the value stored in the variable *cansPerPack*, add 2 to it, and place the result back into *cansPerPack*.

- 1 Compute the value of the right-hand side



- 2 Store the value in the variable

`cansPerPack =` 10

A red arrow points from the value 10 in the previous diagram to the box containing 10 in this diagram, indicating that the result of the computation is being stored back into the variable `cansPerPack`.

Assignment Statement for multiple variables:

```
a, b, c = 5, 3.2, "Hello"
```

```
>>> a
```

```
5
```

```
>>> b
```

```
3.2
```

```
>>> c
```

```
'Hello'
```



- Computers manipulate data values that represent information, and these values can be of different types.
- Each value in a Python program is of a specific type.
- The **data type of a value determines how:**
 - **The data is represented in the computer**
 - **What** operations can be performed on that data.

Integer Data Type (wrote as **int** in Python)

- Represents an **integral value** without a fractional part.
- For example, there must be an integer number of cans in any pack of cans—you cannot have a fraction of a can.

Floating Data Type (wrote as **float** in Python)

- Represents a **fractional part** is required.
- For example, number *0.355 is a floating number.*

String Data Type (wrote as **str** in Python)

- A sequence of characters enclosed in quotation marks “**Hello, World!**” is called a **string**. The message has to be enclosed by either by single (‘...’) or double (“...”)



Table 1 Number Literals in Python

Number	Type	Comment
6	int	An integer has no fractional part.
-6	int	Integers can be negative.
0	int	Zero is an integer.
0.5	float	A number with a fractional part has type float.
1.0	float	An integer with a fractional part .0 has type float.
1E6	float	A number in exponential notation: 1×10^6 or 1000000. Numbers in exponential notation always have type float.
2.96E-2	float	Negative exponent: $2.96 \times 10^{-2} = 2.96 / 100 = 0.0296$
❌ 100,000		Error: Do not use a comma as a decimal separator.
❌ 3 1/2		Error: Do not use fractions; use decimal notation: 3.5.

```
print("Hello") # would print this message Hello
```

A **string** can be stored in a variable

```
greeting = "Hello"
```

```
print(greeting) # would print this message Hello
```



- whose value **should not be changed after it has been assigned an initial value.**
- Python leaves it to the programmer to make sure that constants are not changed
- it is common practice to specify a constant variable with the use of all capital letters for its name.

BOTTLE_VOLUME = 2.0

MAX_SIZE = 100



Input

- When a program asks for user input, it should first print a message that tells user which input is expected. Such a message is called a **prompt**.
- input function can only obtain string of text from user.
- **In Python, displaying** a prompt and reading the keyboard input is combined in one operation. For example:

```
first = input("Enter your first name: ")
```

Enter your first name: █

Enter your first name: Rodolfo█

Input and Output Variables

- To read an integer value, first use the input function to obtain the data as a string, *then convert it to an integer using the int function*. For example:

```
userInput = input("Please enter the number of bottles: ")
```

```
bottles = int(userInput)
```

1 `userInput = input("Please enter the number of bottles: ")`

The prompt is displayed to the program user

2 `userInput = input("Please enter the number of bottles: ")`

userInput = 2 4

The string that the user entered

3 `bottles = int(userInput)`

24

bottles = 24



Input and Output Variables

- To read a floating-point value from the user, the same approach is used, except *the input string has to be converted to a float*.

```
userInput = input("Enter price per bottle: ")  
price = float(userInput)
```

- The previous examples could be written as

```
bottles = int( input("Please enter the number of bottles: ") )  
price = float( input("Enter price per bottle: ") )
```

Output

- Is used to display messages and results to the user.

```
print("Hello, world")    # display  Hello, world
```

```
print(2+3)               # display  5
```

```
print("2+3=", 2+3)      #display   2+3= 5
```

Basic Arithmetic Operators

- Addition $+$ $(5 + 2 = 7)$
- Subtraction $-$ $(5 - 2 = 3)$
- Multiplication $*$ $(5 * 2 = 10)$
- Float Division $/$ $(5 / 2 = 2.5)$
- Integer Division $//$ $(5 // 2 = 2)$ (floor division)
- Power $**$ $(5 ** 2 = 25)$
- Reminder $\%$ $(5 \% 2 = 1)$

Basic Arithmetic operators

$$b \times \left(1 + \frac{r}{100}\right)^n \rightarrow b * (1 + r / 100) ** n$$

Table 3 Floor Division and Remainder

Expression (where n = 1729)	Value	Comment
n % 10	9	For any positive integer n, n % 10 is the last digit of n.
n // 10	172	This is n without the last digit.
n % 100	29	The last two digits of n.
n % 2	1	n % 2 is 0 if n is even, 1 if n is odd (provided n is not negative)
-n // 10	-173	-173 is the largest integer ≤ -172.9 . We will not use floor division for negative numbers in this book.



Operators' precedence

Operator	Description
()	Parenthesis
**	Exponentiation
*, /	Multiplication, Float Division
+, -	Sign positive, sign negative

Operators' precedence

```
a = 20  
b = 10  
c = 15  
d = 5  
e = 0
```

When you execute the above program, it produces the following result

```
Value of (a + b) * c / d is 90  
Value of ((a + b) * c) / d is 90  
Value of (a + b) * (c / d) is 90  
Value of a + (b * c) / d is 50
```

```
e = (a + b) * c / d      #( 30 * 15 ) / 5  
print "Value of (a + b) * c / d is ", e
```

```
e = ((a + b) * c) / d   # (30 * 15 ) / 5  
print "Value of ((a + b) * c) / d is ", e
```

```
e = (a + b) * (c / d);   # (30) * (15/5)  
print "Value of (a + b) * (c / d) is ", e
```

```
e = a + (b * c) / d;     # 20 + (150/5)  
print "Value of a + (b * c) / d is ", e
```



- A **string is a sequence of characters**. For example, the string "Hello" is a sequence of five characters.

`print("Hello")` String is "Hello"

`greeting = "Hello"` String is "Hello"

`print(greeting)` will print "Hello"

- We use double quotation marks "" around strings because this is a common convention in many other programming languages. However, interactive Python interpreter displays strings with single quotation mark ''

```
>>> print("This is a string.", 'So is this.')  
This is a string. So is this.
```

- A **string literal** denotes a particular string (such as "Hello"). In Python,
- String literals are specified by enclosing a sequence of characters within a matching pair of either single or double quotes.

```
>>> message = 'He said "Hello"'
>>> print(message)
He said "Hello"
```

```
>>> question='What is your name?'
>>> print(question)
What is your name?
>>> question="What is your name?"
>>> print(question)
What is your name?
```



- **Length, concatenation and repetition:**

Number of characters in a string is called the *length of the string*. For example, the length of "Harry" is 5

```
length = len("World!")    # length is 6
```

- In Python, use **+** **operator** to **concatenate** two strings. For example.

```
firstName = "Harry"
lastName = "Morgan"
name = firstName + lastName
```

→ results in the string "HarryMorgan"

- You can produce a string that is the result of **repeating a string multiple times**.
 - For example, suppose you need to print a dashed line. Instead of specifying a literal string with 10 dashes, you can use the ***** operator to create a string that is comprised of the string "-" repeated 10 times. For example

```
>>> dashes = "-" * 10
>>> print(dashes)
-----
```

- **Length, concatenation and repetition:**

What if you'd like the first and last name **separated by a space?**

```
name = firstName + " " + lastName
```

This statement concatenates three strings: first Name, the string literal " ", and lastName.

The result is ***Harry Morgan***

- A string of any length can be repeated using the * operator. For example, the statements

```
message = "Echo..."
```

```
print(message * 5)
```

The result is

Echo...Echo...Echo...Echo...Echo...



- Access Characters within Strings

Strings are sequences of **Unicode** characters. You can access the **individual characters of a string based on their position within the string**. This position is called the ***index*** of the character.

W O R D

A string is a sequence of characters.

For example:

```
name = "Harry"
```

the statements

```
first = name[0]
```

```
last = name[4]
```

H	a	r	r	y
0	1	2	3	4

H	a	r	r	y
0	1	2	3	4


```
first = H
```

```
last = y
```

- Python contains a standard library that can be used to create powerful programs.
- A **library is a collection of code that** has been written and translated by someone else, ready for you to use in your program.
- A **standard library is a library that is considered part of the language and must** be included with any Python system.
- Python's standard library is organized into **modules. Related functions and data** types are grouped into the same module.

- Python's math module includes a number of mathematical **functions**. To use any function from this module, you must first **import the function**.
- To include all functions defined in math, write **import math** at the top of the python code.
- Or, to include certain function inside math (as **sqrt**), write **from math import sqrt**
- To **call** sqrt function, write **y = sqrt(x)**

Table 5 Selected Functions in the math Module

Function	Returns
<code>sqrt(x)</code>	The square root of x . ($x \geq 0$)
<code>trunc(x)</code>	Truncates floating-point value x to an integer.
<code>cos(x)</code>	The cosine of x in radians.
<code>sin(x)</code>	The sine of x in radians.
<code>tan(x)</code>	The tangent of x in radians.
<code>exp(x)</code>	e^x
<code>degrees(x)</code>	Convert x radians to degrees (i.e., returns $x \cdot 180/\pi$)
<code>radians(x)</code>	Convert x degrees to radians (i.e., returns $x \cdot \pi/180$)
<code>log(x)</code> <code>log(x, base)</code>	The natural logarithm of x (to base e) or the logarithm of x to the given <i>base</i> .

Table 4 Built-in Mathematical Functions

Function	Returns
<code>abs(x)</code>	The absolute value of x .
<code>round(x)</code> <code>round(x, n)</code>	The floating-point value x rounded to a whole number or to n decimal places.
<code>max(x₁, x₂, ..., x_n)</code>	The largest value from among the arguments.
<code>min(x₁, x₂, ..., x_n)</code>	The smallest value from among the arguments.

Math Library

$$Y = x + 1 * x + 2 * x + 3$$

```
import math
```

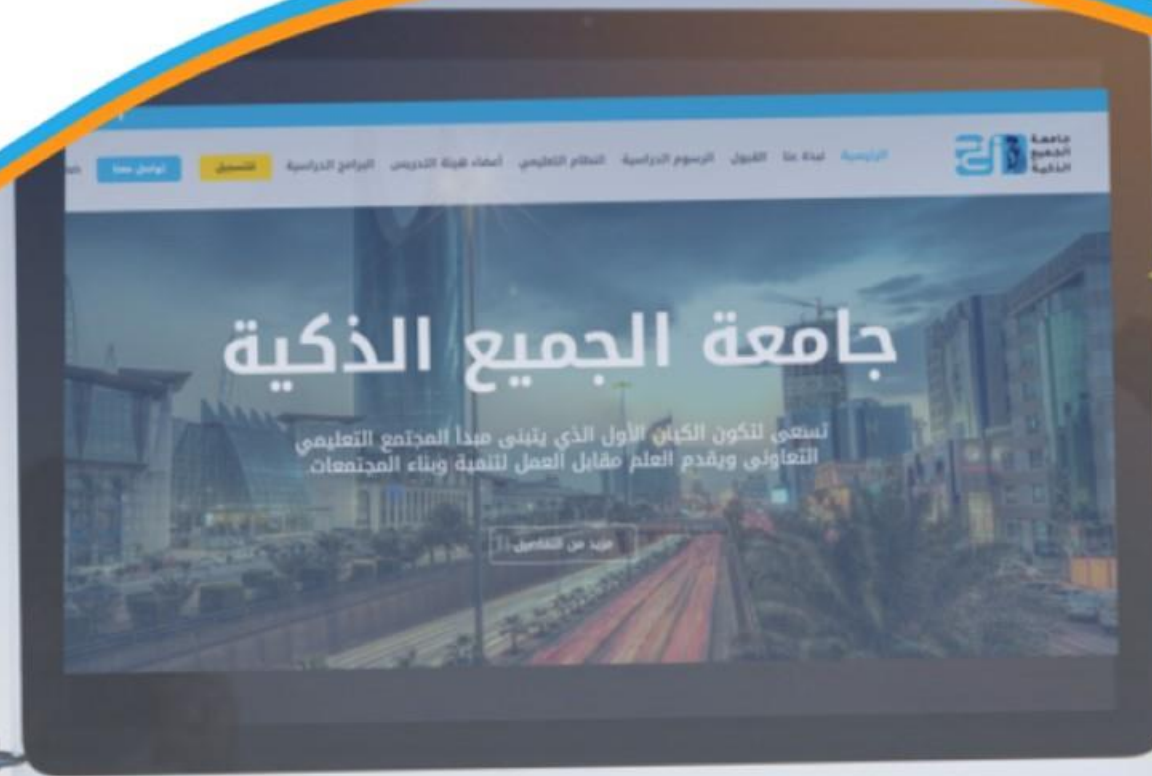
```
x=10
```

```
y=math.sqrt(x+1)* math.sqrt(x+2)* math.sqrt(x+3)
```

```
print("y=",y)
```



جامعة الجميع الذكية
EVERYONE'S SMART UNIVERSITY



Thank you