


College of
Technology
and Design



Python for Data Science

Part 1: Fundamentals

By: Võ Văn Hải
Email: vovanhai@ueh.edu.vn

1

Lecturer information

- ▶ Full name: Võ Văn Hải
- ▶ Email: vovanhai@ueh.edu.vn
- ▶ Zalo: UEH_Python_25D1INF50915201_S7

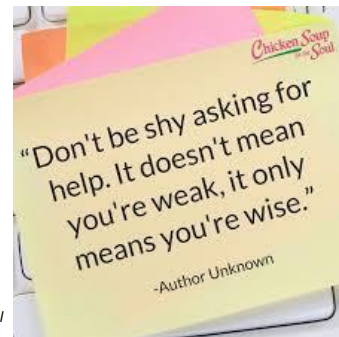
2

Ask

The art and science of asking questions is the source of all knowledge.

- Thomas Berger

- ▶ Do not hesitate to ask!
- ▶ If something is not clear, stop me and ask.
- ▶ During exercises (you can also ask others).

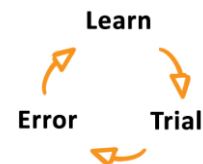


Source: Chicken Soup for the Soul

3

Failure

- ▶ Coding is all about trial and error.
- ▶ Don't be afraid of it.
- ▶ Error messages aren't scary, they are useful.



4

Introduction to



5

5



6

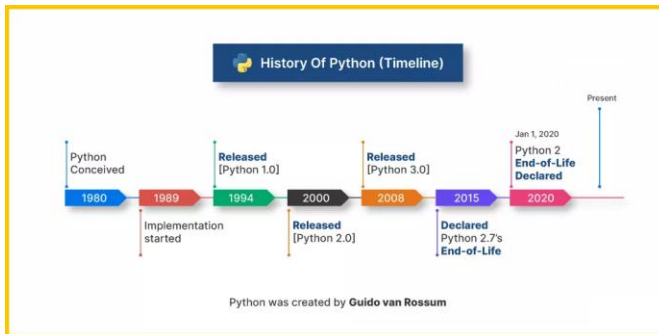
6

History

- ▶ Started by Guido Van Rossum as a hobby
- ▶ Now widely spread
- ▶ Open Source! Free!
- ▶ Versatile



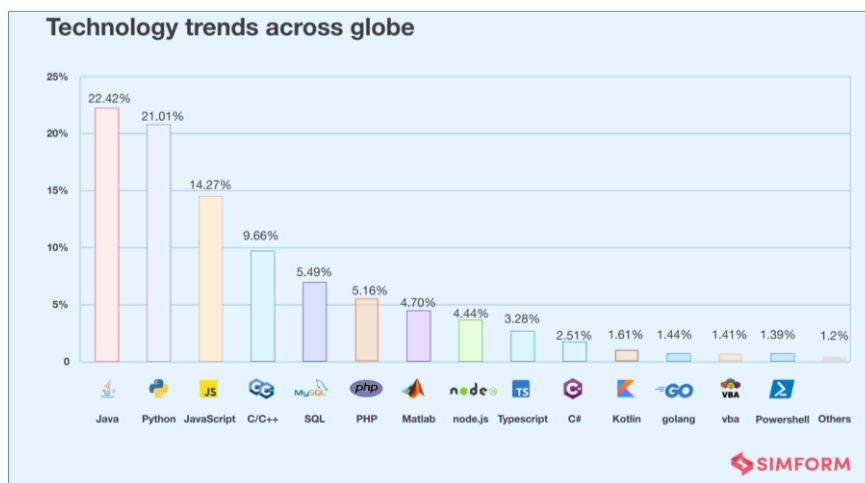
The designer of Python, Guido van Rossum, at OSCON 2006 (src: wikipedia)



Source: unstop.com

7

Programming Languages of 2024

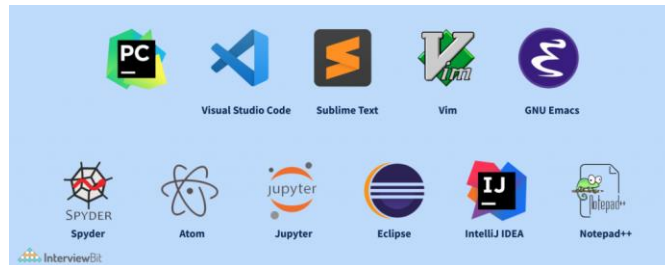


Source: <https://www.simform.com/blog/top-programming-languages/>

8

Start with python

- ▶ Install Python
 - <https://www.python.org/>
- ▶ Python tutorial
 - <https://docs.python.org/3/tutorial/>
 - ...
 - Ask [google](#)
- ▶ IDEs



9

Your First Python Program

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

```
Python 3.11.5 (tags/v3.11.5:cce6ba9, Aug 24 2023, 14:38:34) [MSC v.1936 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> print("Hello, World!")
Hello, World!
>>>
```

```
C:\WINDOWS\system32\cmd.exe - py
C:\Users\VoVanHai>py
Python 3.11.5 (tags/v3.11.5:cce6ba9, Aug 24 2023, 14:38:34) [MSC v.1936 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> print("Hello, World!")
Hello, World!
>>>
```

```
code - eip | src | tam | X.py
Project: C:\Users\VoVanHai\OneDrive\src\COMES
code | X.py | print("Hello, World!")
Run: .py
Hello, World!
```

11

Comment

- ▶ Comments can be used to:
 - explain Python code.
 - make the code more readable.
 - prevent execution when testing code.
- ▶ Comments starts with a `#`, and Python will ignore them
- ▶ Single line comments

```
# This is a comment
print("Hello, World!") # This is a comment
```

- ▶ Multiline Comments

```
"""
This is a comment
written in
more than just one line
"""
print("Hello, World!")
```

www.it-ebooks.info

12

12

Variables

Python as a calculator

- ▶ Let us calculate the distance between Edinburgh and London in km

```
403 * 1.60934
648.56402
```

- ▶ Great calculator but how can we make it store values?
 - Do this by defining variables
- ▶ The value can later be called by the variable name
- ▶ Variable names are case sensitive and unique

```
distanceToLondonMiles = 403
mileToKm = 1.60934
distanceToLondonKm = distanceToLondonMiles * mileToKm
distanceToLondonKm
648.56402
```

The variable *mileToKm* can be used in the next block without having to define it again

www.it-ebooks.info

13

13

Variables

- ▶ Variables are containers for storing data values.
- ▶ Creating Variables
 - Python **has no command for declaring a variable**.
 - A variable is created the moment you first assign a value to it.
- ▶ Variables do not need to be declared with any particular type, and can even change type after they have been set.

```
x = 5
y = "John"
print(x)
print(y)
```

```
x = 4 # x is of type int
x = "Sally" # x is now of type str
print(x)
```

```
# If you want to specify the data type of a variable, this can be done with casting.
x = str(3) # x will be '3'
y = int(3) # y will be 3
z = float(3) # z will be 3.0
```

```
# You can get the data type of a variable with the type() function.
x = 5
y = "John"
print(type(x))
print(type(y))
```

15

Variables

Variable Names

- ▶ A variable can have a short name (like x and y) or a more descriptive name (age, carname, total_volume). Rules for Python variables:
 - A variable name must start with a letter or the underscore character
 - A variable name cannot start with a number
 - A variable name can only contain alpha-numeric characters and underscores (A-z, 0-9, and _)
 - Variable names are case-sensitive (age, Age and AGE are three different variables)
 - A variable name cannot be any of the Python keywords.
- ▶ Variable names with more than one word can be difficult to read. --> several techniques you can use to make them more readable:

Camel Case	Pascal Case	Snake Case
Each word, except the first, starts with a capital letter: <code>myVariableName = "John"</code>	Each word starts with a capital letter: <code>MyVariableName = "John"</code>	Each word is separated by an underscore character: <code>my_variable_name = "John"</code>

16

Variables

Assign Multiple Values

► Many Values to Multiple Variables

- Python allows you to assign values to multiple variables in one line

```
x, y, z = "Orange", "Banana", "Cherry"
```

► One Value to Multiple Variables

- The *same* value can be assigned to multiple variables in one line

```
x = y = z = "Orange"
```

► Unpack a Collection

- If you have a collection of values in a list, tuple etc. Python allows you to extract the values into variables. This is called unpacking.

```
fruits = ["apple", "banana", "cherry"]
x, y, z = fruits
```

17

Input and Output

Output

► Using print() function for print value(s)/variable(s) to standard device

```
print("Hello python's world")
```

```
name, age = "Alice", 30
print("Name:", name, "Age:", age) # ->Name: Alice Age: 30
```

► Output formatting in Python with various techniques including

- the format() method,
- manipulation of the sep and end parameters,
- f-strings, and
- the versatile % operator.

► These methods enable precise control over how data is displayed, enhancing the readability and effectiveness of your Python programs.

```
amount = 150.75
print("Amount:
${:.2f}".format(amount)) # Amount:
$150.75
```

```
# end Parameter with '@'
print("Python", end="@") # Python@is great!
print("is great!")
# Separating with Comma
print('X', 'Y', 'Z', sep='_') # X_Y_Z
# another example
print('vovanhai', 'ueh.edu.vn', sep='@') # vovanhai@ueh.edu.vn
```

18

Input and Output

Output

- Using f-string (discuss later)

```
name, age = 'Teo', 23
print(f'Hello, My name is {name} and I'm {age} years old.')
```

- Using % Operator: We can use '%' operator. % values are replaced with zero or more value of elements. The formatting using % is similar to that of 'printf' in the C programming language.

- %d –integer
- %f – float
- %s – string
- %x –hexadecimal
- %o – octal

```
i, f, h = 10, 13.6, 365
print("i=%d, f=%f, h=%x " % (i, f, h))
# i=10, f=13.600000, h=16d
```

www.it-ebooks.info

19

19

Input and Output

Input

- Python input() function is used to take user input. By default, it returns the user input in form of a string.

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

- Change the Type of Input

```
# Taking input as int
# Typecasting to int
n = int(input("How many roses?: "))
print(n)
```

```
# Taking input from the user
num = int(input("Enter a value: "))
```

- Taking multiple input

```
# taking two inputs at a time
x, y = input("Enter two values: ").split()
print("Number of boys: ", x)
print("Number of girls: ", y)
```

www.it-ebooks.info

20

20

Variables

Exercises

1. Create a variable named `car_name` and assign the value `Volvo` to it.
2. Create a variable named `x` and assign the value `50` to it.
3. Display the sum of `5 + 10`, using two variables: `x` and `y`.
4. Create a variable called `z`, assign `x + y` to it, and display the result.
5. Insert the correct syntax to assign values to multiple variables in one line:

```
x y z = "Orange", "Banana", "Cherry"
```

6. Check the data type of all your variables using `type()` built-in function
7. Run `help('keywords')` in Python shell or in your file to check for the Python reserved words or keywords

Basic Data Types

Data Types

Built-in Data Types

- Variables can store data of different types, and different types can do different things.
- Python has the following data types built-in by default, in these categories:

Category	Type
Text Type:	<code>str</code>
Numeric Types:	<code>int</code> , <code>float</code> , <code>complex</code>
Sequence Types:	<code>list</code> , <code>tuple</code> , <code>range</code>
Mapping Type:	<code>dict</code>
Set Types:	<code>set</code> , <code>frozenset</code>
Boolean Type:	<code>bool</code>
Binary Types:	<code>bytes</code> , <code>bytearray</code> , <code>memoryview</code>
None Type:	<code>NoneType</code>

- Getting the Data Type

```
x = 5
print(type(x)) # result: <class 'int'>
```

https://www.tutorialspoint.com/python/python_data_types.htm

23

23

Data Types

Setting the Data Types

- In Python, the data type is set when you assign a value to a variable:

Example	Data Type
<code>x = "Hello World"</code>	<code>str</code>
<code>x = 20</code>	<code>int</code>
<code>x = 20.5</code>	<code>float</code>
<code>x = 1j</code>	<code>complex</code>
<code>x = ["apple", "banana", "cherry"]</code>	<code>list</code>
<code>x = ("apple", "banana", "cherry")</code>	<code>tuple</code>
<code>x = range(6)</code>	<code>range</code>
<code>x = {"name": "John", "age": 36}</code>	<code>dict</code>
<code>x = {"apple", "banana", "cherry"}</code>	<code>set</code>
<code>x = frozenset({"apple", "banana", "cherry"})</code>	<code>frozenset</code>
<code>x = True</code>	<code>bool</code>
<code>x = b"Hello"</code>	<code>bytes</code>
<code>x = bytearray(5)</code>	<code>bytearray</code>
<code>x = memoryview(bytes(5))</code>	<code>memoryview</code>
<code>x = None</code>	<code>NoneType</code>

https://www.tutorialspoint.com/python/python_data_types.htm

24

24

Data Types

Setting the Specific Data Type

- If you want to specify the data type, you can use the following constructor functions:

Example	Data Type
<code>x = str("Hello World")</code>	str
<code>x = int(20)</code>	int
<code>x = float(20.5)</code>	float
<code>x = complex(1j)</code>	complex
<code>x = list(("apple", "banana", "cherry"))</code>	list
<code>x = tuple(("apple", "banana", "cherry"))</code>	tuple
<code>x = range(6)</code>	range
<code>x = dict(name="John", age=36)</code>	dict
<code>x = set(("apple", "banana", "cherry"))</code>	set
<code>x = frozenset(("apple", "banana", "cherry"))</code>	frozenset
<code>x = bool(5)</code>	bool
<code>x = bytes(5)</code>	bytes
<code>x = bytearray(5)</code>	bytearray
<code>x = memoryview(bytes(5))</code>	memoryview

25

Numbers

- There are three numeric types in Python:

- int
- float
- complex

```
x = 1 # int
y = 2.8 # float
z = 1j # complex
```

```
print(type(x))
print(type(y))
print(type(z))
```

- Variables of numeric types are created when you assign a value to them:

```
# Int
x = 1 # <class 'int'>
y = 35656222554887711
z = -3255522
```

```
# Float
x = 1.10 # <class 'float'>
y = 1.0
z = -35.59
```

```
# Complex
x = 3 + 5j # <class 'complex'>
y = 5j
z = -5j
```

- You can convert from one type to another with the `int()`, `float()`, and `complex()` methods:

```
x = 1 # int
y = 2.8 # float
z = 1j # complex
```

```
# convert from int to float:
a = float(x)
```

```
# convert from float to int:
b = int(y)
```

```
# convert from int to complex:
c = complex(x)
```

Random number

```
import random
print(random.randrange(1, 10))
```

26

Numbers

Specify a Variable Type

- ▶ A variable can be specified with a type by using casting.
- ▶ Casting in python is therefore done using constructor functions:
 - `int()` - constructs an integer number from an integer literal, a float literal (by removing all decimals), or a string literal (providing the string represents a whole number)
 - `float()` - constructs a float number from an integer literal, a float literal or a string literal (providing the string represents a float or an integer)
 - `str()` - constructs a string from a wide variety of data types, including strings, integer literals and float literals
- ▶ Examples

```
x = int(1) # x will be 1
y = int(2.8) # y will be 2
z = int("3") # z will be 3
```

```
x = float(1) # x will be 1.0
y = float(2.8) # y will be 2.8
z = float("3") # z will be 3.0
w = float("4.2") # w will be 4.2
```

```
x = int(1) # x will be 1
y = int(2.8) # y will be 2
z = int("3") # z will be 3
```

27

Numbers

Exercises

1. Declare 5 as num_one and 4 as num_two
 - a) Add num_one and num_two and assign the value to a variable total
 - b) Subtract num_two from num_one and assign the value to a variable diff
 - c) Multiply num_two and num_one and assign the value to a variable product
 - d) Divide num_one by num_two and assign the value to a variable division
 - e) Use modulus division to find num_two divided by num_one and assign the value to a variable remainder
 - f) Calculate num_one to the power of num_two and assign the value to a variable exp
 - g) Find floor division of num_one by num_two and assign the value to a variable floor_division
2. The radius of a circle is 30 meters.
 - a) Calculate the area of a circle and assign the value to a variable name of area_of_circle
 - b) Calculate the circumference of a circle and assign the value to a variable name of circum_of_circle
 - c) Take radius as user input and calculate the area.

28

Strings

- Strings in python are surrounded by either *single* quotation marks, or *double* quotation marks.

- 'hello' is the same as "hello".

- It is possible to use quotes inside a string, if they don't match the quotes surrounding the string:

```
print("It's alright")
print("He is called 'Johnny'")
print('He is called "Johnny"')
```

- You can assign a multiline string to a variable by using three quotes:

```
a = """Lorem ipsum dolor sit amet,
ut labore et dolore magna aliqua."""
```

```
a = "Lorem ipsum dolor sit amet,
ut labore et dolore magna aliqua."
```

- Strings in Python are arrays of bytes representing unicode characters. The *square brackets []* can be used to *access elements* of the string.

```
a = "Hello, World!"
print(a[1])
```

```
# Loop through the letters in a string
for x in "life is a present":
    print(x)
```

- To get the length of a string, use the `len()` function.

```
a = "Hello, World!"
print(len(a))
```

```
x = 'life is a present'
for i in range(len(x)):
    print(x[i])
```

29

Strings

Slicing Strings

- You can return a range of characters by using the slice syntax.
 - Specify the start index and the end index, separated by a colon, to return a part of the string.

```
# Get the characters from position 2 to position 5 (not included):
b = "Hello, World!"
print(b[2:5]) # --> llo
```

- By leaving out the start index, the range will start at the first character:

```
# Get the characters from the start to position 5 (not included):
b = "Hello, World!"
print(b[:5]) # Hello
```

- By leaving out the end index, the range will go to the end:

```
# Get the characters from position 2, and all the way to the end:
b = "Hello, World!"
print(b[2:]) # llo, World!
```

- Use negative indexes to start the slice from the end of the string:

```
# From: "o" in "World!" (position -5)
# To, but not included: "d" in "World!" (position -2):
b = "Hello, World!"
print(b[-5:-2]) # orl
```

30

Strings

Modify Strings

- ▶ The `upper()` and `lower()` methods return the string in upper case and lower case:

```
a = "Hello, World!"
print(a.upper())
print(a.lower())
```

- ▶ The `strip()` method removes any whitespace from the beginning or the end:

```
a = "  Hello, World!  "
print(a.strip()) # returns "Hello, World!"
```

- ▶ The `replace()` method replaces a string with another string:

```
a = "Hello, World!"
print(a.replace("H", "J")) # Jello, World!
```

- ▶ The `split()` method returns a list where the text between the specified separator becomes the list items

```
a = "Hello, World!"
print(a.split(",")) # returns ['Hello', ' World!']
```

- ▶ Python String Methods

Read more: <https://www.digitalocean.com/community/tutorials/python-string-functions>

31

31

Strings

String Format

- ▶ we cannot combine strings and numbers like this:

```
age = 36
txt = "My name is John, I am " + age
print(txt)
```



```
Traceback (most recent call last):
  File "X.py", line 2, in <module>
    txt = "My name is John, I am " + age
TypeError: can only concatenate str (not "int") to str
```

- ▶ The F-String can be used to format string with placeholders

- To specify a string as an f-string, simply put an `f` in front of the string literal and add curly brackets `{}` as placeholders for variables and other operations.
- A placeholder can contain variables, operations, functions, and modifiers to format the value.

```
price = 59
txt = f"The price is {price} dollars" # -> The price is 59 dollars
```

- It can be used for formatting the number

```
price = 59.678
txt = f"The price is {price:.2f} dollars" # -> The price is 59.68 dollars
```

- Or, using with an expression

```
# Return "Expensive" if the price is over 50, otherwise return "Cheap":
price = 49
txt = f"It is very {'Expensive' if price > 50 else 'Cheap'}
```

```
txt = f"The price is {13 * 59} dollars" # The price is 767 dollars
```

32

32

Strings

String Format (continue)

► Formatting types

```
price = 59000
txt = f"The price is {price:,} dollars" # -> The price is 59,000 dollars
```

:<	Left aligns the result (within the available space)
:>	Right aligns the result (within the available space)
:^	Center aligns the result (within the available space)
:=	Places the sign to the left most position
:+	Use a plus sign to indicate if the result is positive or negative
:-	Use a minus sign for negative values only
:	Use a space to insert an extra space before positive numbers (and a minus sign before negative numbers)
;	Use a comma as a thousand separator
_	Use an underscore as a thousand separator
:b	Binary format
:c	Converts the value into the corresponding Unicode character
:d	Decimal format
:e	Scientific format, with a lower-case e
:E	Scientific format, with an upper-case E
:f	Fix point number format
:F	Fix point number format, in uppercase format (show inf and nan as INF and NAN)
:g	General format
:G	General format (using a upper case E for scientific notations)
:o	Octal format
:x	Hex format, lower case
:X	Hex format, upper case
:n	Number format
:%	Percentage format

33

33

Strings

String Format (continue)

- The format() method can be used to format strings (before v3.6), but f-strings are faster and the preferred way to format strings.

```
quantity = 3
item_no = 567
price = 49
myorder = "I want {} pieces of item number {} for {:.2f} dollars."
print(myorder.format(quantity, item_no, price))
# -> I want 3 pieces of item number 567 for 49.00 dollars.
```

- You can use index numbers :

```
age, name = 36, "John"
txt = "His name is {1}. {1} is {0} years old."
print(txt.format(age, name)) # His name is John. John is 36 years old.
```

- You can also use named indexes:

```
my_order = "I have a {car_name}, it is a {model}."
print(my_order.format(car_name="Ford", model="Mustang"))
# -> I have a Ford, it is a Mustang.
```

34

34

Strings

Escape characters

- ▶ To insert characters that are illegal in a string, use an escape character.
- ▶ An escape character is a backslash \ followed by the character you want to insert.

txt = "We are the so-called \"Vikings\" from the north."

Code	Result
\'	Single Quote
\\	Backslash
\n	New Line
\r	Carriage Return
\t	Tab
\b	Backspace
\f	Form Feed
\ooo	Octal value
\xhh	Hex value

35

Strings

Exercises

Give: x = "Hello World"

1. Use the *len* function to print the length of the string.
2. Get the characters from index 2 to index 4 (llo).
3. Return the string without any whitespace at the beginning or the end.
4. Convert the value of txt to upper/lower case.
5. Replace the character l with a L.
6. Enter a string prompt keyboard then display it reverse order

36

Booleans

- ▶ In programming you often need to know if an expression is True or False.
 - You can evaluate any expression in Python, and get one of two answers, True or False.
 - When you compare two values, the expression is evaluated, and Python returns the Boolean answer:

```
print(10 > 9) #->True
print(10 == 9) #->False
print(10 < 9) #->False
```

- ▶ The bool() function allows you to evaluate any value and give you True or False in return. Almost any value is evaluated to True if it has some sort of content.

- Any string is True, except empty strings.
- Any number is True, except 0.
- Any list, tuple, set, and dictionary are True, except empty ones.

```
bool("abc")
bool(123)
bool(["apple", "cherry", "banana"])
```

- ▶ There are not many values that evaluate to False, except empty values, such as (), [], {}, "", the number 0, and the value None.

- The value False evaluates to False.

```
bool(False)
bool(None)
bool(0)
```

```
bool("")
bool(())
bool([])
bool({})
```

www.tutorialspoint.com/python/boolean.htm

37

37

Operators

38

Operators

- Operators are used to perform operations on variables and values.
- Python divides the operators in the following groups:

- Arithmetic operators
- Assignment operators
- Comparison operators
- Logical operators
- Identity operators
- Membership operators
- Bitwise operators

Operator	Name	Example
+	Addition	x + y
-	Subtraction	x - y
*	Multiplication	x * y
/	Division	x / y
%	Modulus	x % y
**	Exponentiation	x ** y
//	Floor division	x // y

Arithmetic Operators

Operator	Name	Example
==	Equal	x == y
!=	Not equal	x != y
>	Greater than	x > y
<	Less than	x < y
>=	Greater than or equal to	x >= y
<=	Less than or equal to	x <= y

Comparison Operators

39

Operators

Assignment Operators

Operator	Example	Same As
=	x = 5	x = 5
+=	x += 3	x = x + 3
-=	x -= 3	x = x - 3
*=	x *= 3	x = x * 3
/=	x /= 3	x = x / 3
%=	x %= 3	x = x % 3
//=	x //= 3	x = x // 3
**=	x **= 3	x = x ** 3
&=	x &= 3	x = x & 3
=	x = 3	x = x 3
^=	x ^= 3	x = x ^ 3
>>=	x >>= 3	x = x >> 3
<<=	x <<= 3	x = x << 3
:=	print(x := 3)	x = 3 print(x)

Membership Operators

Operator	Description	Example
in	Returns True if a sequence with the specified value is present in the object	x in y
not in	Returns True if a sequence with the specified value is not present in the object	x not in y

Logical Operators

Operator	Description	Example
and	Returns True if both statements are true	x < 5 and x < 10
or	Returns True if one of the statements is true	x < 5 or x < 4
not	Reverse the result, returns False if the result is true	not(x < 5 and x < 10)

Identity Operators

Operator	Description	Example
is	Returns True if both variables are the same object	x is y
is not	Returns True if both variables are not the same object	x is not y

40

Operators

Bitwise Operators

Operator	Name	Description	Example
&	AND	Sets each bit to 1 if both bits are 1	x & y
	OR	Sets each bit to 1 if one of two bits is 1	x y
^	XOR	Sets each bit to 1 if only one of two bits is 1	x ^ y
~	NOT	Inverts all the bits	~x
<<	Zero fill left shift	Shift left by pushing zeros in from the right and let the leftmost bits fall off	x << 2
>>	Signed right shift	Shift right by pushing copies of the leftmost bit in from the left, and let the rightmost bits fall off	x >> 2

www.howtogeek.com

41

41

Operators

Operator Precedence

- ▶ Operator precedence describes the order in which operations are performed.
- ▶ The precedence order is described in the table below, starting with the highest precedence at the top:

Operator	Description
()	Parentheses
**	Exponentiation
+x -x ~x	Unary plus, unary minus, and bitwise NOT
* / // %	Multiplication, division, floor division, and modulus
+ -	Addition and subtraction
<< >>	Bitwise left and right shifts
&	Bitwise AND
^	Bitwise XOR
	Bitwise OR
== != > >= < <= is is not in not in	Comparisons, identity, and membership operators
not	Logical NOT
and	AND
or	OR

www.howtogeek.com

42

42

Operators

Exercises

1. Write a program that prompts the user to enter base and height of the triangle and calculate an area of this triangle (area = $0.5 \times b \times h$).
2. Write a script that prompts the user to enter side a, side b, and side c of the triangle. Calculate the perimeter of the triangle (perimeter = $a + b + c$).
3. Get length and width of a rectangle using prompt. Calculate its area (area = length \times width) and perimeter (perimeter = $2 \times (\text{length} + \text{width})$).
4. Get radius of a circle using prompt. Calculate the area (area = $\pi \times r \times r$) and circumference ($c = 2 \times \pi \times r$) where $\pi = 3.14$.
5. Calculate the slope, x-intercept and y-intercept of $y = 2x - 2$ $y = \frac{1}{2}x + 1$
6. Slope is ($m = \frac{y_2 - y_1}{x_2 - x_1}$). Find the slope and Euclidean distance between point (2, 2) and point (6, 10).
7. Compare the slopes in tasks 5 and 6.
8. Calculate the value of y ($y = x^2 + 6x + 9$). Try to use different x values and figure out at what x value y is going to be 0.
9. Find the length of 'python' and 'dragon' and make a falsy comparison statement.

43

43

Operators

Exercises

10. Use and operator to check if 'on' is found in both 'python' and 'dragon'
11. I hope this course is not full of jargon. Use in operator to check if jargon is in the sentence.
12. There is no 'on' in both dragon and python
13. Find the length of the text python and convert the value to float and convert it to string
14. Even numbers are divisible by 2 and the remainder is zero. How do you check if a number is even or not using python?
15. Write a script that prompts the user to enter hours and rate per hour. Calculate pay of the person?
16. Write a script that prompts the user to enter number of years. Calculate the number of seconds a person can live. Assume a person can live hundred years

44

44

Python Flow Control

45

Conditional Statements

Introduction

- ▶ In computer programming, the if statement is a conditional statement. It is used to execute a block of code only when a specific condition is met.
- ▶ Types of Control Flow in Python
 - Python If Statement
 - Python If Else Statement
 - Python Nested If Statement
 - Python Elif
 - Ternary Statement (Short Hand If Else Statement)

46

46

Python if...else Statement

Python if Statement - Example

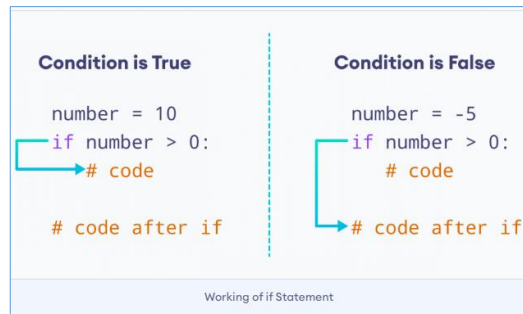
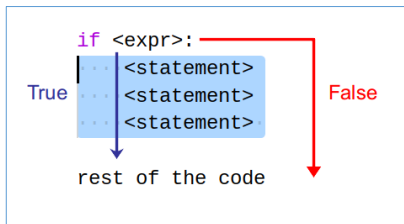
► Syntax

```
if (condition):
    # code for execution
    # when the condition's result is True
```

```
number = int(input('Enter a number: '))

# check if number is greater than 0
if number > 0:
    print(f'{number} is a positive number.')

print('A statement outside the if statement.')
```



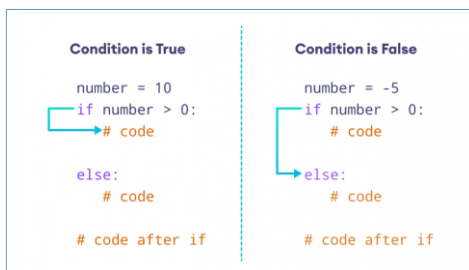
47

Python if...else Statement

Python if...else Statement

► Syntax

```
if condition:
    # body of if statement
else:
    # body of else statement
```



```
number = int(input('Enter a number: '))
if number > 0:
    print('Positive number')
else:
    print('Not a positive number')

print('This statement always executes')
```

48

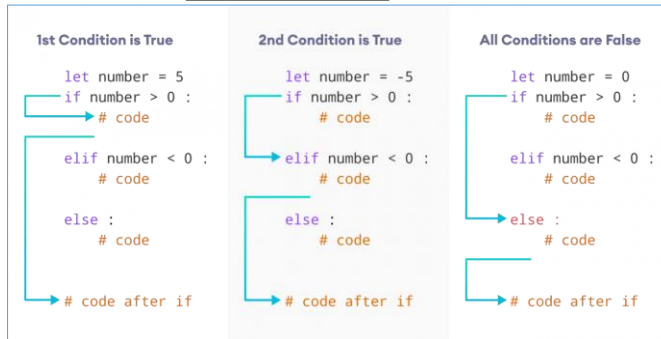
Python if...else Statement

The if...elif...else Statement

► Syntax

```
if condition1:
    # code block 1
elif condition2:
    # code block 2
else:
    # code block 3
```

```
number = -5
if number > 0:
    print('Positive number')
elif number < 0:
    print('Negative number')
else:
    print('Zero')
print('This statement is always executed')
```



www.btech.edu.in

49

49

Python if...else Statement

Nested if Statements

```
number = 5
# outer if statement
if number >= 0:
    # inner if statement
    if number == 0:
        print('Number is 0')
    # inner else statement
    else:
        print('Number is positive')
# outer else statement
else:
    print('Number is negative')
```

Outer if Condition is True

```
number = 5
if number >= 0:
    if number == 0:
        #code
    else:
        #code
else:
    #code
```

Outer if Condition is False

```
number = -5
if number >= 0:
    if number == 0:
        #code
    else:
        #code
else:
    #code
```

www.btech.edu.in

50

50

Python if...else Statement

Compact if (Ternary If Else) Statement

- ▶ If you have only one statement to execute, one for if, and one for else, you can put it all on the same line:

- ▶ Syntax:

```
do_work if condition else do_other_work
```

- ▶ Example

```
a = 30
b = 330
print("A") if a > b else print("B")
print("A") if a > b else print("=") if a == b else print("B")
```

www.it-ebooks.info

51

51

Python if...else Statement

Exercises

1. Write a program to check a person is eligible for voting or not (saccept age from user)
2. Write a program to check whether a number entered by user is even or odd.
3. Write a program to check whether a number is divisible by 7 or not.
4. Write a program to check the last digit od a number (entered by user) is divisible by 3 or not.
5. Write a Python program to guess a number between 1 and 9.
6. Write a program to accept a number from 1 to 7 and display the name of the day like 1 for Sunday, 2 for Monday and so on.

www.it-ebooks.info

52

52

The For loop Statement

53

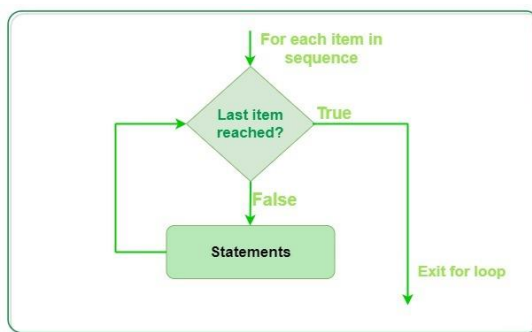
For loop Statement

- ▶ A for loop is used for iterating over a sequence (that is either a list, a tuple, a dictionary, a set, or a string).

- ▶ Syntax

for var **in** sequence:
statements

The for loop iterates over the elements of sequence in order. In each iteration, the body of the loop is executed.



```
languages = ['Swift', 'Python', 'Go']
# access elements of the list one by one
for lang in languages:
    print(lang)
```

```
language = 'Python'
# iterate over each character in language
for x in language:
    print(x)
```

```
# iterate from i = 0 to i = 3
for i in range(4):
    print(i)
```

54

For loop Statement

Python for loop with else clause

- ▶ A for loop can have an optional else clause. This else clause executes after the iteration completes.

```
digits = [0, 1, 5]
for i in digits:
    print(i)
else:
    print("No items left.")
```

- ▶ Note: The else block will not execute if the for loop is stopped by a break statement.

- ▶ Python for loop in One Line

```
Numbers = [x for x in range(11)]
print(Numbers)
```

www.tutorialspoint.com

55

55

For loop Statement

Nested for loops

- ▶ A for loop can also have another for loop inside it.
- ▶ For each cycle of the outer loop, the inner loop completes its entire sequence of iterations.

```
# outer loop
for i in range(2):
    # inner loop
    for j in range(2):
        print(f"i = {i}, j = {j}")
```

```
i = 0, j = 0
i = 0, j = 1
i = 1, j = 0
i = 1, j = 1
```

www.tutorialspoint.com

56

56

For loop Statement

Control Statements with For Loop

► Continue in Python For Loop

- Python continue Statement returns the control to the beginning of the loop.

```
for letter in 'university of economics HCMC':
    if letter == 'e' or letter == 's':
        continue
    print('Current Letter :', letter)
```

► Break in Python For Loop

```
for letter in 'University of economics HCMC':
    # break the loop as soon it sees 'e' or 's'
    if letter == 'e' or letter == 's':
        break
    print('Current Letter :', letter) # --> e
```

► For Loop in Python with Pass Statement

```
# An empty loop
for letter in 'University of economics HCMC':
    pass
print('Last Letter :', letter)
```

57

For loop Statement

Exercises

1. write a program that prints numbers from 1 to 10
2. Write a program to calculate the sum of numbers in a range from 1 to n (n is entered from the keyboard)
3. Write a program to calculate the sum of even (/odd) numbers in a range from 1 to n (n is entered from the keyboard)
4. Write a program to check how many vowels are in a string entered from the keyboard.
5. Write a program to count the number of words in a sentence the user enters.
6. Write a program that implements a game as the following description:
7. 1. The computer generates a random number from 1 to 100
8. 2. The user was asked to guess
9. 3. match the user-guessing number to the generated number
10. The user can only guess five times.

58

Python while Loop

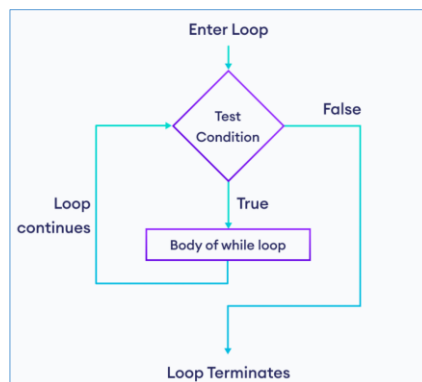
59

Python while Loop

- ▶ In Python, a while loop can be used to repeat a block of code until a certain condition is met.

- ▶ Syntax:

```
while condition:
    # body of while loop
```



```
# Print numbers until the user enters 0
number = int(input('Enter a number: '))
```

```
# iterate until the user enters 0
```

```
while number != 0:
    print(f'You entered {number}.')
    number = int(input('Enter a number: '))

print('The end.')
```

60

60

Python while Loop

Infinite while Loop

- If the condition of a while loop always evaluates to True, the loop runs continuously, forming an *infinite* while loop.

```
age = 28
# the test condition is always True
while age > 19:
    print('Infinite Loop')
```

```
# Initialize a counter
count = 0
# Loop infinitely
while True:
    # Increment the counter
    count += 1
    print(f"Count is {count}")
    # Check if the counter has reached a certain value
    if count == 10:
        # If so, exit the loop
        break
# This will be executed after the loop exits
print("The loop has ended.")
```

www.it-ebooks.info

61

61

Python while Loop

Control Statements in Python while loop

- Python **Continue** Statement returns the control to the beginning of the loop.
- Python **Break** Statement brings control out of the loop.
- The Python **pass** statement to write empty loops. Pass is also used for empty control statements, functions, and classes.

```
# Prints all letters except 'e' and 's'
i = 0
a = 'University of Economics HCMC'
while i < len(a):
    if a[i] == 'e' or a[i] == 's':
        i += 1
        continue
    print('Current Letter :', a[i])
    i += 1
```

break

break the loop as soon it sees 'e' or 's' if we replace the continue with break statement

```
# An empty loop
i = 0
a = 'University of Economics HCMC'
while i < len(a):
    i += 1
    pass
```

www.it-ebooks.info

62

62

Python while Loop

Exercises

1. Guess The Number Game:

- we will generate a random number with the help of randint() function from 1 to 100 and ask the user to guess it.
- After every guess, the user will be told if the number is above or below the randomly generated number.
- The user will win if they guess the number maximum five attempts.
- Ask the user to stop or continue playing again.

2. Write a game that simulate rolling a pair of dice.

- If the sum of two faces is greater than 5 → "Tài"
- Otherwise → "Xỉu"
- User ask for guessing "Tài" or "Xỉu"
- Match the results
- After one turn, ask user for continue playing game.
- **** Extend the game by asking the user to enter an amount of money, then continue playing until the user runs out of money or the user stops playing. Statistics of results.