# Tổng quan

Python được Guido van Rossum phát triển vào cuối những năm 80 và đầu những năm 90 tại Viện Nghiên cứu Quốc gia về Toán học và Khoa học Máy tính ở Hà Lan.

Python có nguồn gốc từ nhiều ngôn ngữ khác, bao gồm ABC, Modula-3, C, C++, Algol-68, SmallTalk, và Unix shell và các ngôn ngữ kịch bản khác.

Python đã được đăng ký bản quyền. Giống như Perl, mã nguồn Python hiện có sẵn theo Giấy phép Công cộng GNU (GPL).

Python hiện được duy trì bởi một nhóm phát triển cốt lõi tại viện, mặc dù Guido van Rossum vẫn giữ một vai trò quan trọng trong việc chỉ đạo tiến trình của nó.

**Tính năng Python**

**Dễ học** − Python có ít từ khóa, cấu trúc đơn giản và cú pháp được xác định rõ ràng. Điều này cho phép học sinh tiếp thu ngôn ngữ một cách nhanh chóng.

**Dễ đọc** – Mã Python được xác định rõ ràng hơn và dễ nhìn hơn.

**Dễ bảo trì** − Mã nguồn của Python khá dễ bảo trì.

**Một thư viện tiêu chuẩn rộng rãi** - Phần lớn thư viện của Python rất di động và tương thích đa nền tảng trên UNIX, Windows và Macintosh.

**Chế độ tương tác** - Python có hỗ trợ chế độ tương tác cho phép thử nghiệm tương tác và gỡ lỗi các đoạn mã.

**Di động** − Python có thể chạy trên nhiều nền tảng phần cứng khác nhau và có giao diện giống nhau trên tất cả các nền tảng.

**Có thể mở rộng** - Bạn có thể thêm các mô-đun cấp thấp vào trình thông dịch Python. Các mô-đun này cho phép các lập trình viên thêm vào hoặc tùy chỉnh các công cụ của họ để hiệu quả hơn.

**Cơ sở dữ liệu** - Python cung cấp giao diện cho tất cả các cơ sở dữ liệu thương mại chính.

**Lập trình GUI** − Python hỗ trợ các ứng dụng GUI có thể được tạo và chuyển sang nhiều lệnh gọi hệ thống, thư viện và hệ thống cửa sổ, chẳng hạn như Windows MFC, Macintosh và hệ thống X Window của Unix.

**Có thể mở rộng** − Python cung cấp cấu trúc và hỗ trợ tốt hơn cho các chương trình lớn so với shell scripting.

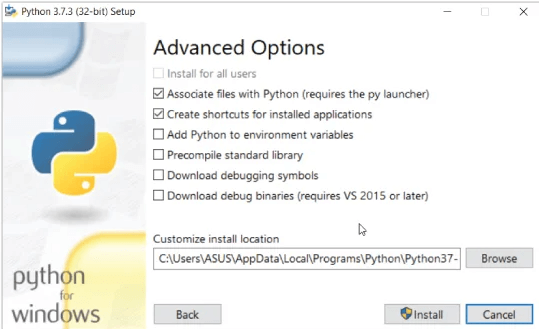
# Thiết lập môi trường

## Cài đặt python

**Bước 1:** Tải Python tại đây: https://www.python.org/downloads/, chọn phiên bản bạn cần.

**Bước 2:** Chạy tệp trình cài đặt và làm theo các bước để cài đặt Python. Trong quá trình cài đặt, hãy chọn Add Python to environment variables. Thao tác này sẽ thêm Python vào các biến môi trường và bạn có thể chạy Python từ bất kỳ phần nào của máy tính.

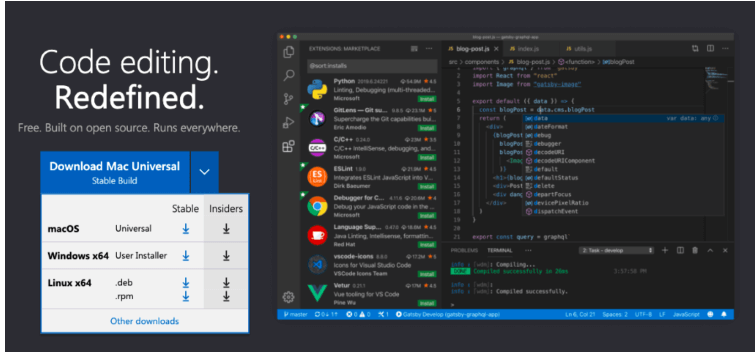
Ngoài ra, bạn có thể chọn đường dẫn nơi Python sẽ được cài đặt.



Sau khi hoàn tất quá trình cài đặt, bạn có thể chạy Python.

## Cài đặt Visual Studio Code

**Bước 1:** Truy cập link của nhà phát hành: <https://code.visualstudio.com/>

****

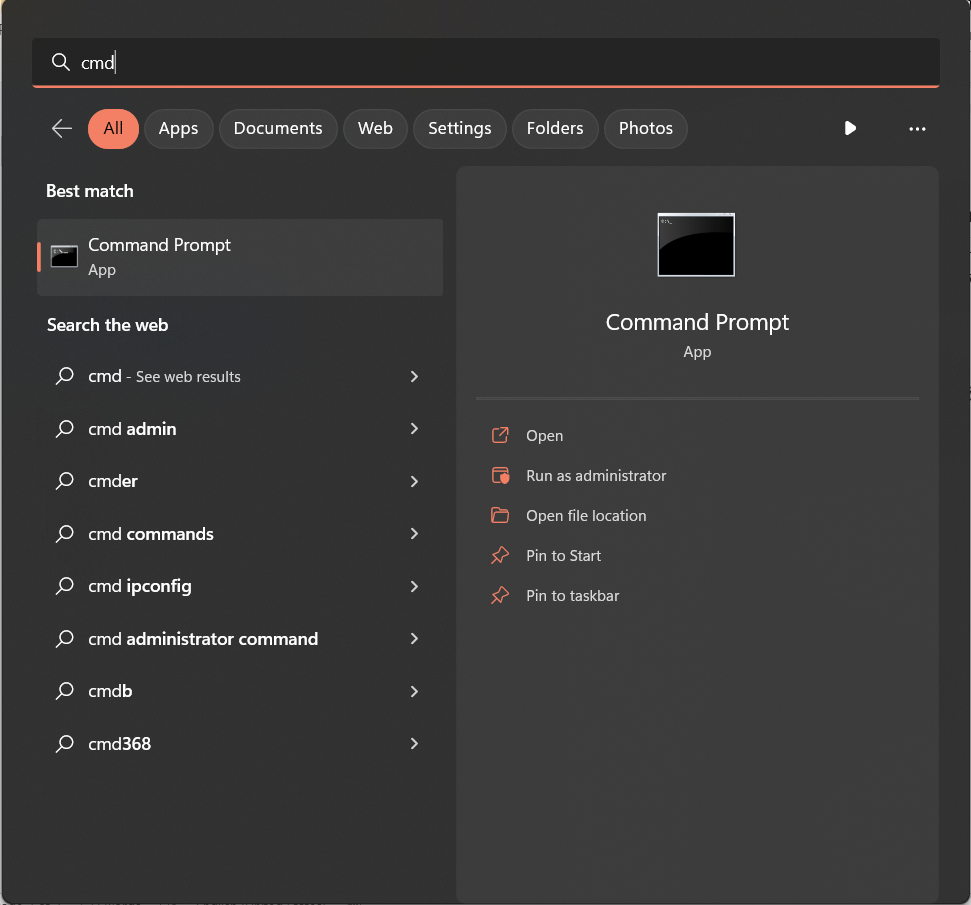
Website sẽ tự Recommend phiên bản thích hợp với máy của bạn, hoặc nếu muốn cài bản khác, bạn chọn trong danh sách như sau:

Sau khi tải xong mở file và tiến hành cài đặt, Chỉ cần Next cho đến khi Finish là xong.

# Cú pháp cơ bản

**Python - Lập trình chế độ tương tác**

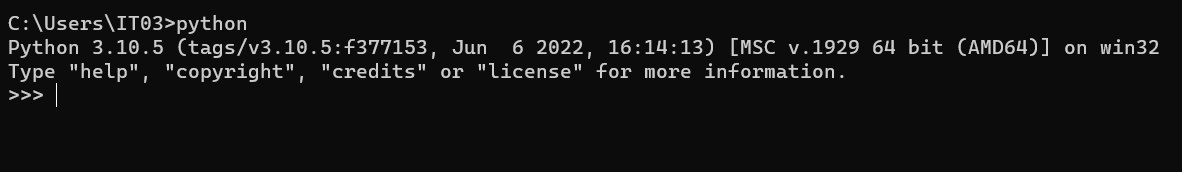
Chúng ta có thể gọi trình thông dịch Python từ dòng lệnh bằng cách gõ python tại dấu nhắc lệnh như sau. Bấm phím bắt win và gõ cmd và chọn open



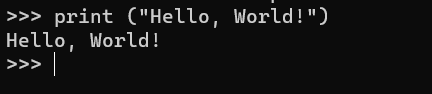
Sẽ hiển thị ra cửa sổ trình thông dịch **command line**



Gõ python sẽ hiển thị.



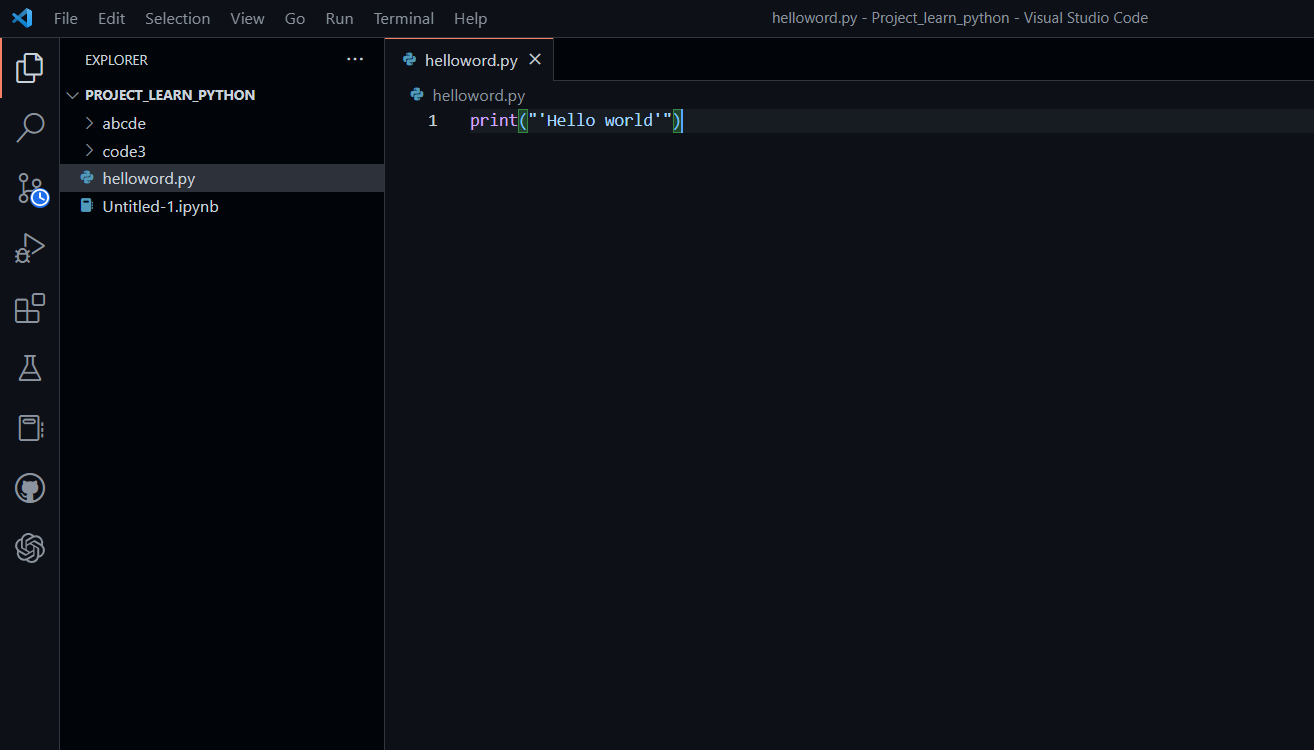
Ở đây >>> biểu thị Dấu nhắc lệnh Python có thể nhập lệnh của mình. Hãy nhập văn bản sau tại dấu nhắc Python và nhấn Enter



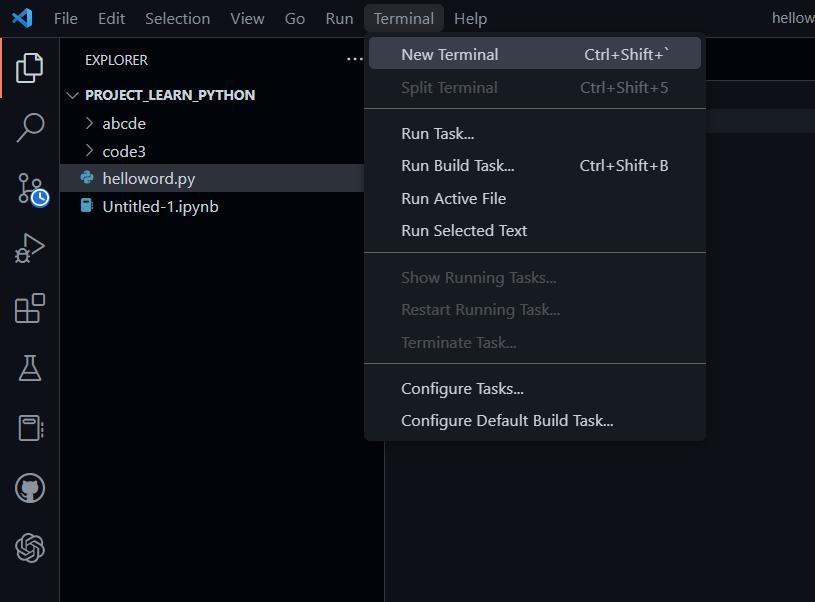
**Python - Lập trình chế độ tập lệnh.**

**Bước 1:** Mở Visual Studio Code.

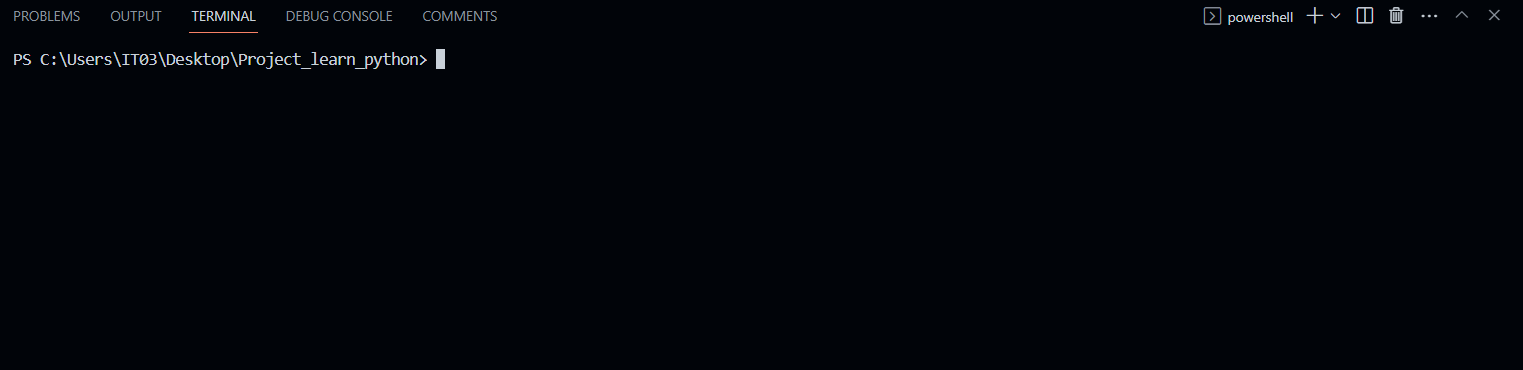
**Bước 2:** chọn open Folder và chon đến nơi lưu file helloword.py



Tạo file hellowrord.py có nội dung như sau. print 'Hello world'.

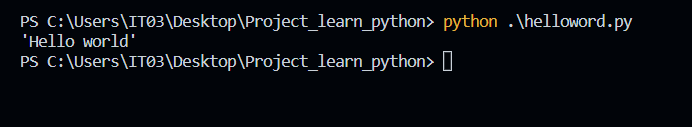
Ở đây lênh print là lệnh cơ bản nhất dùng để xuát giá trị ra màn hình command line 

Sau đó chọn Terminal 🡺 new Terminal sẽ hiển thị giao diện như sau.



Ở đây chúng chung ta đang ở chế đọ lện powershell

Tiếp theo gõ lệnh python helloword.py để chạy chường trình trong file helloword.py



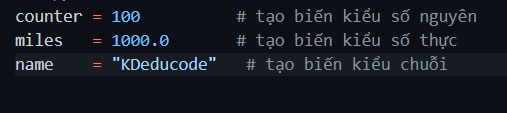
# Biến

Các biến Python là các vị trí bộ nhớ dành riêng được sử dụng để lưu trữ các giá trị trong Chương trình Python. Điều này có nghĩa là khi bạn tạo một biến, bạn dành một khoảng trống trong bộ nhớ. Dựa trên kiểu dữ liệu của một biến, trình thông dịch Python phân bổ bộ nhớ và quyết định những gì có thể được lưu trữ trong bộ nhớ dành riêng. Do đó, bằng cách gán các kiểu dữ liệu khác nhau cho các biến Python, bạn có thể lưu trữ số nguyên, số thập phân hoặc ký tự trong các biến này.

**Tạo biến Python**

Các biến Python không cần khai báo rõ ràng để dành dung lượng bộ nhớ hoặc bạn có thể nói để tạo một biến. Biến Python được tạo tự động khi bạn gán giá trị cho nó. Dấu bằng (=) dùng để gán giá trị cho biến.

Toán hạng bên trái của toán tử = là tên của biến và toán hạng bên phải của toán tử = là giá trị được lưu trữ trong biến. Ví dụ

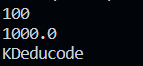


**In biến Python**

****

Khi chúng ta tạo một biến Python và gán giá trị cho nó, chúng ta có thể in nó bằng hàm print(). Sau đây là phần mở rộng của ví dụ trước và chỉ ra cách in các biến khác nhau trong Python:

Ở đây, 100, 1000.0 và " KDeducode" lần lượt là các giá trị được gán cho các biến *counter* , *mile* và *name* . Khi chạy chương trình Python ở trên, điều này tạo ra kết quả sau.



**Xóa một biến**

Bạn có thể xóa tham chiếu đến một đối tượng số bằng cách sử dụng câu lệnh del. Cú pháp của câu lệnh del là:

del var1[,var2[,var3[....,varN]]]]

Bạn có thể xóa một đối tượng hoặc nhiều đối tượng bằng cách sử dụng câu lệnh del. Ví dụ

var\_a = var\_b = var = 1

del var

del var\_a, var\_b

**Ví dụ**

Các ví dụ sau cho thấy cách chúng ta có thể xóa một biến và nếu chúng ta cố gắng sử dụng một biến đã xóa thì trình thông dịch Python sẽ báo lỗi:

counter = 100

print (counter)

del counter

print (counter)

Điều này sẽ tạo ra kết quả sau:

100 và thông báo lỗi

**Traceback (most recent call last):**

**File "main.py", line 7, in <module>**

**print (counter)**

**NameError: name 'counter' is not defined**

Python cho phép bạn gán một giá trị cho nhiều biến đồng thời, điều đó có nghĩa là bạn có thể tạo nhiều biến cùng một lúc.

**Ví dụ**

a = b = c = 100

print (b)

print (c)

Điều này tạo ra kết quả sau:



Ở đây, một đối tượng số nguyên được tạo với giá trị 1 và cả ba biến được gán cho cùng một vị trí bộ nhớ. Bạn cũng có thể gán nhiều đối tượng cho nhiều biến. Ví dụ -

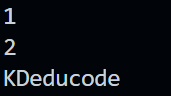
a,b,c = 1,2,"KDeducode"

print (a)

print (b)

print (c)

Điều này tạo ra kết quả sau:



Ở đây, hai đối tượng số nguyên có giá trị 1 và 2 lần lượt được gán cho các biến a và b và một đối tượng chuỗi có giá trị " KDeducode " được gán cho biến c.

**Tên biến Python**

Mỗi biến Python nên có một tên duy nhất như a, b, c. Tên biến có thể có ý nghĩa như màu sắc, tuổi, tên, v.v. Có một số quy tắc nhất định cần được lưu ý khi đặt tên biến Python:

* Tên biến phải bắt đầu bằng một chữ cái hoặc ký tự gạch dưới
* Tên biến không được bắt đầu bằng số hoặc bất kỳ ký tự đặc biệt nào như $, (, \* %, v.v.
* Tên biến chỉ có thể chứa các ký tự chữ và số và dấu gạch dưới (Az, 0-9 và \_ )
* Tên biến Python phân biệt chữ hoa chữ thường, có nghĩa là Tên và TÊN là hai biến khác nhau trong Python.
* Không thể sử dụng các từ khóa dành riêng cho Python để đặt tên cho biến.

**Ví dụ**

Sau đây là các tên biến Python hợp lệ:

counter = 100

\_count  = 100

name1 = "KD"

name2 = "KD"

Age  = 20

kd = 100000

print (counter)

print (\_count)

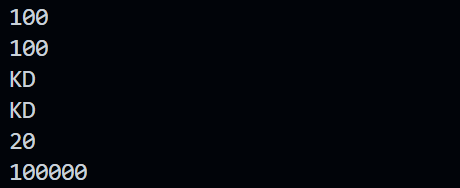
print (name1)

print (name2)

print (Age)

print (kd)

Điều này sẽ tạo ra kết quả sau:

****

**Ví dụ**

Sau đây là các tên biến Python không hợp lệ:

*1counter* = 100

*$*\_count  = 100

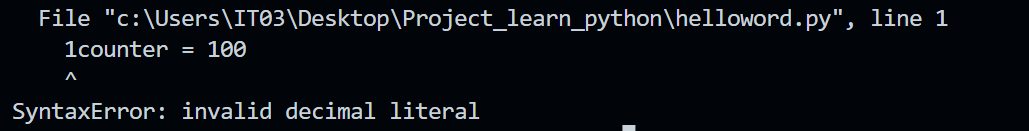
KD-salary = 100000

print (*1counter*)

print (*$*count)

print (KD-salary)

Điều này sẽ tạo ra kết quả sau:



**Biến cục bộ Python**

Các biến cục bộ trong Python được định nghĩa bên trong một hàm. Chúng ta không thể truy cập biến bên ngoài chức năng.

Hàm Python là một đoạn mã có thể tái sử dụng và bạn sẽ tìm hiểu thêm về hàm trong hướng dẫn [Python - Hàm](https://www.tutorialspoint.com/python/python_functions.htm).

Sau đây là một ví dụ để hiển thị việc sử dụng các biến cục bộ:

def sum(x,y):

   sum = x + y

   return sum

print(sum(5, 10)) # 🡺 kết quả là 15

**Biến toàn cục Python**

Bất kỳ biến nào được tạo bên ngoài một hàm đều có thể được truy cập trong bất kỳ hàm nào và vì vậy chúng có phạm vi toàn cục. Sau đây là một ví dụ về biến toàn cục:

x = 5

y = 10

def sum():

   sum = x + y

   return sum

print(sum()) # 🡺 kết quả là 15

# Kiểu dữ liệu

Các kiểu dữ liệu Python được sử dụng để xác định loại biến. Nó xác định loại dữ liệu chúng ta sẽ lưu trữ trong một biến. Dữ liệu được lưu trữ trong bộ nhớ có thể thuộc nhiều loại. Ví dụ: tuổi của một người được lưu dưới dạng giá trị số và địa chỉ của người đó được lưu dưới dạng ký tự chữ và số.

Python có nhiều kiểu dữ liệu tích hợp sẵn mà chúng ta sẽ thảo luận trong hướng dẫn này:

* Dữ liệu kiểu số - int, float, complex
* Dữ liệu kiểu chuỗi - str
* Dữ liệu mang tính liên tục - list, tuple, range
* Dữ liệu kiểu nhị phân - bytes, bytearray, memoryview
* Dữ liệu kiểu từ điển - dict
* Dữ liệu kiểu đúng/sai - bool
* Kiểu dữ liệu bộ - set, frozenset
* Không có kiểu dữ liệu- NoneType

Kiểu dữ liệu số Python

var1 = 1

var2 = 10

var3 = 10.023

Python hỗ trợ bốn loại số khác nhau

* int (số nguyên có dấu)
* long (số nguyên dài, chúng cũng có thể được biểu diễn dưới dạng bát phân và thập lục phân)
* float (giá trị thực dấu phẩy động)
* số phức (số phức)

**Ví dụ**

a=100  # biến số nguyên.

print("Kiểu biến có giá trị", a, " là ", type(a))

b=20.345 # biến float.

print("Kiểu biến có giá trị", b, " là ", type(b))

c=10+3j # biến phức tạp.

print("Kiểu biến có giá trị", c, " là ", type(c))

d = "Kdeducode" # biến kiểu chuỗi

print("Kiểu biến có giá trị", d, " là ", type(d))oán tử

# Câu lệnh if ... Else

**Khái Niệm:**

Câu lệnh if là câu lệnh ra quyết định đơn giản nhất. Nó được sử dụng để quyết định xem một câu lệnh hoặc một khối câu lệnh nào đó sẽ được thực thi hay không. Tức là nếu một điều kiện nào đó là đúng thì một khối câu lệnh sẽ được thực thi, ngược lại thì khối câu lệnh đó sẽ không được thực thi.

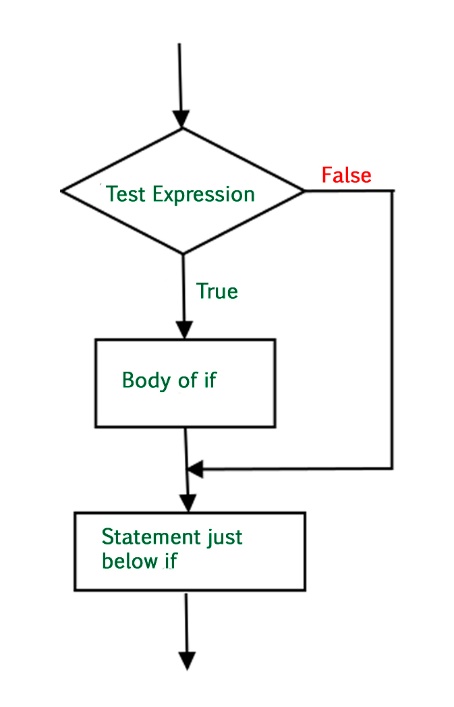
if test expression:

    statement(s)

Ở đây, chương trình chỉ đánh giá (các) câu lệnh test expression và sẽ chỉ thực thi nếu biểu thức được kiểm tra cho kết quả là True.

Nếu biểu thức kiểm tra là False, (các) câu lệnh sẽ không được thực thi. Trong Python, phần nội dung của câu lệnh if được biểu thị bằng thụt đầu dòng. Phần nội dung bắt đầu bằng một dấu thụt đầu dòng và dòng không có dấu thụt lề sẽ được coi là phần cuối. Python diễn giải các giá trị khác 0 là True. None và 0 được hiểu là False.

Sơ đồ câu lệnh if



**Ví Dụ Về Câu Lệnh if**

num = 3

if num > 0:

    print(num, "là số lớn hơn 0")

print("KDeducode")

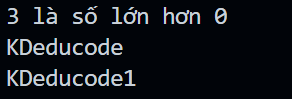
num = -1

if num > 0:

    print(num, "là số nhỏ hơn 0")

print("KDeducode1")

**kết quả:**

****

Trong ví dụ trên, num > 0 là biểu thức kiểm tra.

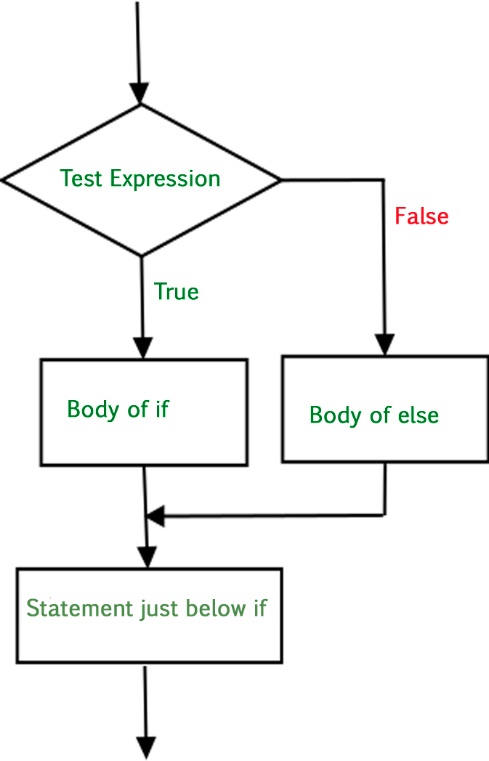
Phần thân của if chỉ được thực thi nếu điều này được đánh giá là True.

Khi biến num bằng 3, biểu thức kiểm tra là true và các câu lệnh bên trong phần thân của if được thực thi.

Nếu biến num bằng -1, biểu thức kiểm tra là sai và các câu lệnh bên trong phần thân của if bị bỏ qua.

Câu print() lệnh nằm ngoài khối if (không có liên kết). Do đó, nó được thực thi bất kể biểu thức kiểm tra.

Sơ đồ câu lệnh if … else



Ví dụ về câu lệnh if...else

i = 20

if (i < 15):

    print ("i nhỏ hơn 15")

else:

    print ("i lớn hơn 15")

print ("bên ngoài khối lệnh if else")

"""

kết quả:

i lớn hơn 15

bên ngoài khối lệnh if else

"""

Khối mã theo sau câu lệnh else được thực thi vì điều kiện hiện tại trong câu lệnh if là sai sau khi gọi câu lệnh không nằm trong khối (không có dấu cách).

# Vòng lặp for và while

Duyệt qua danh sách Vòng lặp **for** được sử dụng để lặp qua một chuỗi (đó là danh sách, bộ dữ liệu, từ điển, tập hợp hoặc chuỗi).

Điều này ít giống với từ khóa for trong các ngôn ngữ lập trình khác và hoạt động giống một phương thức lặp hơn như được tìm thấy trong các ngôn ngữ lập trình hướng đối tượng khác.

Với vòng lặp for, chúng ta có thể thực thi một tập hợp các câu lệnh, một lần cho mỗi mục trong danh sách, bộ dữ liệu, bộ, v.v.

**Ví dụ**

traiCay = ["Cam", "Chuối", "Xoài","Táo","Ổi"]

for x in traiCay:

  print(x)

"""

kết quả:

Cam

Chuối

Xoài

Táo

Ổi

"""

Duyêt qua chuỗi:

for x in "Trái cây":

  print(x)

"""

kết quả:

T

r

á

i

c

â

y"""

Với câu lệnh break, chúng ta có thể dừng vòng lặp trước khi nó lặp qua tất cả các mục:

traiCay = ["Cam", "Chuối", "Xoài","Táo","Ổi"]

for x in traiCay:

  if x == "Cam":

    print(x)

    break

"""

Kết quả:

Cam

"""

Với câu lệnh continue, chúng ta có thể dừng vòng lặp hiện tại của vòng lặp và tiếp tục với bước tiếp theo:

traiCay = ["Cam", "Chuối", "Xoài","Táo","Ổi"]

for x in traiCay:

  if x == "Cam":

    continue

  print(x)

"""

Kết quả:

Chuối

Xoài

Táo

Ổi

"""

**while** trong Python được dùng để lặp lại một khối lệnh hay đoạn code khi điều kiện đúng. Bạn có thể dùng while trong nhiều trường hợp nhưng nó thường được sử dụng khi người dùng không đoán trước được số lần cần lặp lại.

Cú pháp của while trong Python.

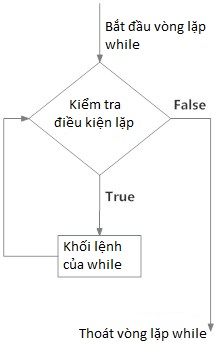
while dieu\_kien:

    Khối lệnh của while

Trong vòng lặp while, dieu\_kien sẽ được kiểm tra đầu tiên, nếu nó là True, thì khối lệnh của vòng lặp sẽ được thực thi. Sau một lần lặp, dieu\_kien sẽ được kiểm tra lại và quá trình lặp này sẽ chỉ dừng cho đến khi điều kiện là False.

Trong Python mọi giá trị khác 0 đều là True, None và 0 được hiểu là False. Đặc điểm này có thể dẫn đến trường hợp là while có thể không chạy vì ngay lần lặp đầu tiên dieu\_kien đã False. Khi đó, khối lệnh của while sẽ bị bỏ qua và phần code dưới khối lệnh while sẽ được thực thi.

Sơ đồ câu lệnh while



Giống như if hay vòng lặp for, khối lệnh của while cũng được xác định thông qua thụt lề. Khối lệnh bắt đầu với thụt lề đầu tiên và kết thúc với dòng không thụt lề đầu tiên liền sau khối.

Ví dụ

count = 1

n = 0

while (n < 8):

      print ('Số thứ', count,' là:', n)

      n = n + 1

      count = count + 1

print ("Hết rồi!")

"""

Kết quả:

Số thứ 1  là: 0

Số thứ 2  là: 1

Số thứ 3  là: 2

Số thứ 4  là: 3

Số thứ 5  là: 4

Số thứ 6  là: 5

Số thứ 7  là: 6

Số thứ 8  là: 7

Hết rồi

"""

Lưu ý :

* Hãy nhớ tăng biến điều kiện trong while (trong ví dụ trên là n), nếu không vòng lặp sẽ trở thành vòng lặp vô hạn - tiếp tục lặp mãi mãi.
* Vòng lặp while yêu cầu biến trong điều kiện phải là giá trị xác định, trong ví dụ trên biến lập chỉ mục lặp là biến n, chúng ta phải đặt giá trị ban đầu cho nó là 1.

Lệnh break trong while

i = 1

while i < 6:

  print(i)

  if i == 3: #kiểm tra điều kiện xem i bằng 3 hay chưa

    break

  i += 1 #cập nhật biến đếm

"""

Kết quả:

1

2

3

"""

Lệnh continue trong while

i = 0

while i < 6:

  i += 1

  if i == 3:

    continue

  print(i)

"""

Kết quả:

1

2

4

5

6

"""

# Hàm

**Hàm** là một nhóm các lệnh có liên quan đến nhau được dùng để thực hiện một tác vụ, nhiệm vụ cụ thể nào đó. Hàm giúp chia chương trình Python thành những khối/phần/mô-đun nhỏ hơn. Khi chương trình Python quá lớn, hoặc cần mở rộng, thì các hàm giúp chương trình có tổ chức và dễ quản lý hơn.

Cú pháp:

def ten\_ham(các tham số/đối số):

 """Chuỗi văn bản để mô tả cho hàm (docstring)"""

 Các câu lệnh

Về cơ bản, một định nghĩa hàm Python sẽ bao gồm các thành phần sau:

1. **Từ khóa def**: Đánh dáu sự bắt đầu của tiêu đề hàm.
2. **ten\_ham**: Là định danh duy nhất dành cho hàm. Việc đặt tên hàm phải tuân thủ theo quy tắc viết tên và định danh trong Python.
3. **Các tham số/đối số**: Chúng ta truyền giá trị cho hàm thông qua các tham số này. Chúng là tùy chọn.
4. **Dấu hai chấm (:)**: Đánh dấu sự kết thúc của tiêu đề hàm.
5. **docstring**: Chuỗi văn bản tùy chọn để mô tả chức năng của hàm.
6. **Các câu lệnh**: Một hoặc nhiều lệnh Python hợp lệ tạo thành khối lệnh. Các lệnh này phải có cùng một mức thụt đầu dòng (thường là 4 khoảng trắng).
7. **Lệnh return**: Lệnh này là tùy chọn, dùng khi cần trả về giá trị từ hàm

# Modules

Module được sử dụng để phân loại code thành các phần nhỏ hơn liên quan với nhau. Hay nói cách khác, Module giúp bạn tổ chức Python code một cách logic để giúp bạn dễ dàng hiểu và sử dụng code đó hơn. Trong Python, Module là một đối tượng với các thuộc tính mà bạn có thể đặt tên tùy ý và bạn có thể gắn kết và tham chiếu.

Về cơ bản, một Module là một file, trong đó các lớp, hàm và biến được định nghĩa. Tất nhiên, một Module cũng có thể bao gồm code có thể chạy.

Để import một Module, bạn có thể sử dụng một trong ba cách dưới đây:

1. import module1[, module2[,... moduleN]
2. from modname import name1[, name2[, ... nameN]]
3. from modname import \*

# Tương tác file

File hay còn gọi là tệp, tập tin. File là tập hợp của các thông tin được đặt tên và lưu trữ trên bộ nhớ máy tính như đĩa cứng, đĩa mềm, CD, DVD ,...

Khi muốn đọc hoặc ghi file, chúng ta cần phải mở file trước. Khi hoàn thành, file cần phải được đóng lại để các tài nguyên được gắn với file được giải phóng.

Do đó, trong Python, một thao tác với file diễn ra theo thứ tự sau.

1. Mở tệp tin
2. Đọc hoặc ghi
3. Đóng tệp

**Mở File trong Python**

Trong Python, có một hàm được xây dựng sẵn phục vụ cho việc mở file: open(). Hàm này trả về đối tượng file hay còn gọi là “handle” vì bạn có thể thực hiện các hoạt động đọc, ghi, sửa đổi trên file đó.

f = open("test.txt")

Có thể xác định cách thức mà tập tin được mở ra để làm gì như read, write, append,... Đây là thông số tùy chọn có thể có hoặc không. Ngoài ra bạn cũng có thể định rõ file mở ra dạng văn bản hay dạng nhị phân.

Chế độ truy cập file mặc định là read (r). Khi dùng mode này chúng ta sẽ nhận được giá trị chuỗi trả về dạng văn bản.

Mặt khác nếu giá trị trả về ở dạng byte thì tệp được mở ra là hình ảnh hoặc exe.

Dưới đây là danh sách các chế độ mode khác nhau khi mở một file:

|  |  |
| --- | --- |
| **MODE** | **MÔ TẢ** |
| ‘r’ | Chế độ chỉ được phép đọc. |
| ‘r+’ | Chế độ được phép đọc và ghi |
| ‘rb’ | Mở file chế độ đọc cho định dạng nhị phân. Con trỏ tại phần bắt đầu của file |
| ‘rb+’ ‘r+b’ | Mở file để đọc và ghi trong định dạng nhị phân. Con trỏ tại phần bắt đầu của file |
| ‘w’ | Mở file để ghi. Nếu file không tồn tại thì sẽ tạo mới file và ghi nội dung, nếu file đã tồn tại thì sẽ bị cắt bớt (truncate) và ghi đè lên nội dung cũ |
| ‘w+’ | Mở file để đọc và ghi. Nếu file không tồn tại thì sẽ tạo mới file và ghi nội dung, nếu file đã tồn tại thì sẽ bị cắt bớt (truncate) và ghi đè lên nội dung cũ |
| ‘wb’ | Mở file để ghi cho dạng nhị phân. Nếu file không tồn tại thì sẽ tạo mới file và ghi nội dung, nếu file đã tồn tại thì sẽ bị cắt bớt (truncate) và ghi đè lên nội dung cũ |
| ‘wb+’ ‘w+b’ | Mở file để đọc và ghi cho dạng nhị phân. Nếu file không tồn tại thì sẽ tạo mới file và ghi nội dung, nếu file đã tồn tại thì sẽ bị cắt bớt (truncate) và ghi đè lên nội dung cũ |
| ‘a’ | Mở file chế độ ghi tiếp. Nếu file đã tồn tại rồi thì nó sẽ ghi tiếp nội dung vào cuối file, nếu file không tồn tại thì tạo một file mới và ghi nội dung vào đó. |
| ‘a+’ | Mở file chế độ đọc và ghi tiếp. Nếu file đã tồn tại rồi thì nó sẽ ghi tiếp nội dung vào cuối file, nếu file không tồn tại thì tạo một file mới và ghi nội dung vào đó. |
| ‘ab’, | Mở file chế độ ghi tiếp ở dạng nhị phân. Nếu file đã tồn tại rồi thì nó sẽ ghi tiếp nội dung vào cuối file, nếu file không tồn tại thì tạo một file mới và ghi nội dung vào đó. |
| ‘ab+’ ‘a+b | Mở file chế độ đọc và ghi tiếp ở dạng nhị phân. Nếu file đã tồn tại rồi thì nó sẽ ghi tiếp nội dung vào cuối file, nếu file không tồn tại thì tạo một file mới và ghi nội dung vào đó. |
| ‘x’ | Mở file chế độ ghi. Tạo file độc quyền mới (exclusive creation) và ghi nội dung, nếu file đã tồn tại thì chương trình sẽ báo lỗi |
| ‘x+’ | Mở file chế độ đọc và ghi. Tạo file độc quyền mới (exclusive creation) và ghi nội dung, nếu file đã tồn tại thì chương trình sẽ báo lỗi |
| ‘xb’ | Mở file chế độ ghi dạng nhị phân. Tạo file độc quyền mới và ghi nội dung, nếu file đã tồn tại thì chương trình sẽ báo lỗi |
| ‘xb+’ ‘x+b’ | Mở file chế độ đọc và ghi dạng nhị phân. Tạo file độc quyền mới và ghi nội dung, nếu file đã tồn tại thì chương trình sẽ báo lỗi |
| ‘b’ | Mở file ở chế độ nhị phân |
| ‘t’ | Mở file ở chế độ văn bản (mặc định) |

**Đóng File trong Python**

Sau khi thực hiện xong các thao tác với file thì bạn cần đóng nó lại.

Đóng file để đảm bảo quy chế đóng mở và giải phóng bộ nhớ cho chương trình nên điều này là cần thiết.

Việc đóng file được xây dựng trong Python bằng hàm close().

Python cũng tự động đóng một file khi đối tượng tham chiếu của file đã được tái gán cho một file khác. Tuy nhiên, sử dụng phương thức close() để đóng một file vẫn tốt hơn.

f = open("test.txt",encoding = 'utf-8')# thực hiện các thao tác với tệpf.close()

Tuy nhiên cách này chưa thực sự đảm bảo. Vẫn có trường hợp một số ngoại lệ xảy ra khi chúng ta thực hiện các thao tác với file khiến chương trình tự động thoát ra mà không đóng tệp.

Để đảm bảo hơn, bạn nên sử dụng khối lệnh try...finally (finally sẽ luôn luôn được thực thi bất chấp có hay không ngoại lệ) ở đây.

try: f = open("test.txt",encoding = 'utf-8') # thực hiện các thao tác với tệpfinally: f.close()

Bằng cách này, ta có thể yên tâm file được đóng đúng ngay cả khi phát sinh ngoại lệ khiến chương trình dừng đột ngột.

Một cách khác để đóng file là sử dụng câu lệnh with. Lệnh with cho chúng ta bảo đảm rằng file luôn luôn được đóng mà không cần biết những logic xử lý bên trong.

try: f = open("test.txt",encoding = 'utf-8') # thực hiện các thao tác với tệpfinally: f.close()

So sánh hai cách viết này thì chúng ta đã thấy rất rõ ràng rằng, sử dụng with cho chúng ta cách viết code ngắn gọn hơn hẳn.

**Ghi File trong Python**

Để ghi một file ta cần mở file bằng cú pháp để ghi, sử dụng mode write ‘w’, append ‘a’ hoặc mode tạo độc quyền ‘x’

Bạn cần cẩn thận với chế độ ‘w’, vì nó ghi đè lên nội dung nếu file đã tồn tại, các dữ liệu trước đó sẽ bị xóa.

Nếu bạn ghi vào file dạng nhị phân các chuỗi văn bản hoặc chuỗi dạng byte thì kết quả trả về sẽ là số kí tự được ghi vào file.

with open("test.txt",'w',encoding = 'utf-8') as f: f.write("KDEducode\n") f.write("Kiến thức - Kinh nghiệm - Hỏi đáp\n\n") f.write("KDEducode.vn\n")

kết quả:

Educode Kiến thức - Kinh nghiệm - Hỏi KDEducode.vn

**Đọc File**

Tương tự ghi file, để đọc một file ta cần mở file bằng cú pháp để đọc, sử dụng mode read ‘r’.

Dùng read(size)

Sử dụng phương thức read(size) để lấy về dữ liệu có kích thước bằng size. Nếu để trống tham số này thì nó sẽ đọc hết file hoặc nếu file quá lớn thì nó sẽ đọc đến khi giới hạn của bộ nhớ cho phép.

f = open("test.txt",'r',encoding = 'utf-8')

a = f.read(12) # đọc 12 kí tự đầu tiên

print('Nội dung 11 kí tự đầu là:\n', (a))

b = f.read(35) # đọc 35 kí tự tiếp theo

print('Nội dung 35 kí tự tiếp theo là:\n', (b))

c = f.read() # đọc phần còn lại

print('Nội dung phần còn lại là:\n', (c))

**Dùng tell() và seek()**

Ngoài ra, ta có phương thức tell() cho bạn biết vị trí hiện tại bên trong file. Nói cách khác, việc đọc và ghi tiếp theo sẽ diễn ra tiếp tục trên các byte đó.

Phương thức seek() thay đổi vị trí hiện tại bên trong file.

f = open("test.txt",'r',encoding = 'utf-8')

a = f.read(12) # đọc 12 kí tự đầu tiên

print('Nội dung là: \n', (a))

b = f.tell() # Kiểm tra vị trí hiện tại

print ('Vị trí hiện tại: ', (b))

f.seek(0) # Đặt lại vị trí con trỏ tại vị trí đầu

c = f.read()

print('Nội dung mới là: \n', (c))

**Dùng readline()**

Phương thức này cho phép đọc từng dòng trong file:

f = open("test.txt",'r',encoding = 'utf-8')

a = f.readline()

print ('Nội dung dòng đầu: ', (a))

b = f.readline()

print ('Nội dung dòng 2: ', (b))

c = f.readline()

print ('Nội dung dòng 3: ', (c))

d = f.readline()

print ('Nội dung dòng 4: ', (d))

**Dùng readlines()**

Phương thức readlines() trả về toàn bộ các dòng còn lại trong file và trả về giá trị rỗng khi kết thúc file.

f = open("test.txt",'r',encoding = 'utf-8')

a = f.readline()

print ('Nội dung dòng đầu: ', (a))

b = f.readlines()

print ('Nội dung các dòng còn lại: \n', (b))

c = f.readlines()

print ('Nội dung các dòng còn lại: \n', (c))

**Một số phương thức làm việc với File trong Python**

|  |  |
| --- | --- |
| **PHƯƠNG THỨC** | **MÔ TẢ** |
| close() | Đóng một file đang mở. Nó không thực thi được nếu tập tin đã bị đóng. |
| fileno() | Trả về một số nguyên mô tả file (file descriptor). |
| flush() | Xóa sạch bộ nhớ đệm của luồng file. |
| isatty() | Trả về TRUE nếu file được kết nối với một thiết bị đầu cuối. |
| read(n) | Đọc n kí tự trong file. |
| readable() | Trả về TRUE nếu file có thể đọc được. |
| readline(n=-1) | Đọc và trả về một dòng từ file. Đọc nhiều nhất n byte/ký tự nếu được chỉ định. |
| readlines(n=-1) | Đọc và trả về một danh sách các dòng từ file. Đọc nhiều nhất n byte/ký tự nếu được chỉ định. |
| seek(offset,from=SEEK\_SET) | Thay đổi vị trí hiện tại bên trong file. |
| seekable() | Trả về TRUE nếu luồng hỗ trợ truy cập ngẫu nhiên. |
| tell() | Trả về vị trí hiện tại bên trong file. |
| truncate(size=None) | Cắt gọn kích cỡ file thành kích cỡ tham số size. |
| writable() | Trả về TRUE nếu file có thể ghi được. |
| write(s) | Ghi s kí tự vào trong file và trả về. |
| writelines(lines) | Ghi một danh sách các dòng và file. |

# Hướng đối tượng

# Xử lý ngoại lệ