### **Python là gì?**

  
**Python** là một ngôn ngữ lập trình cấp cao sử dụng các hướng dẫn để dạy máy tính cách thực hiện một tác vụ.

#### Python là một ngôn ngữ lập trình dễ học, mạnh mẽ.

#### Một ngôn ngữ gần gũi hơn với ngôn ngữ của con người (như tiếng Anh) được gọi là ngôn ngữ cấp cao.

#### Python cung cấp một cách tiếp cận dễ dàng để lập trình hướng đối tượng.

#### Hướng đối tượng là một cách tiếp cận được sử dụng để viết chương trình.

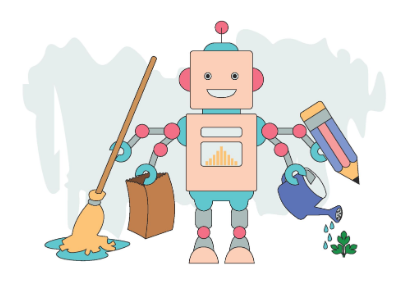
#### Python là một ngôn ngữ mã nguồn mở và miễn phí.

#### Đó là, bạn có thể đọc, sửa đổi và phân phối mã nguồn của các tập lệnh Python.

#### Nó được phát triển bởi **Guido van Rossum** và được phát hành vào năm 1991.

#### Chúng ta sẽ tìm hiểu về Object Oriented Programming sau.

### **Python được sử dụng ở đâu?**

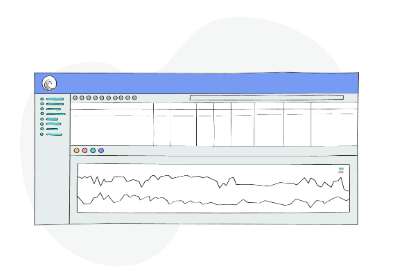
  
Python tìm thấy ứng dụng của nó trong các lĩnh vực khác nhau.

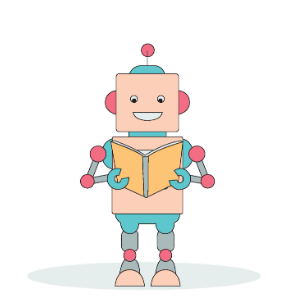
#### Hãy xem một số tình huống cụ thể trong đó Python được sử dụng.



#### Python được sử dụng để tạo **các ứng dụng web**.

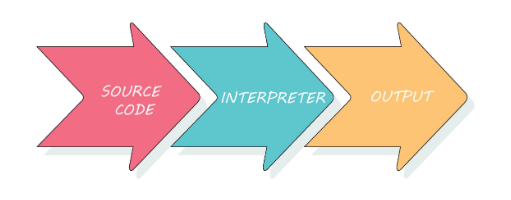
  
Python được sử dụng trong **phát triển trò chơi**.

  
Python cũng được sử dụng để tạo **các ứng dụng máy tính để bàn**.



#### Python cũng tìm thấy các ứng dụng của nó trong **Machine learning và Data science**.

### **Python hoạt động như thế nào?**



* Đầu tiên, chúng tôi viết hướng dẫn bằng ngôn ngữ Python dựa trên tiếng Anh.
* Python là một ngôn ngữ thông dịch, vì vậy không cần phải biên dịch chúng.
* Các chương trình Python chạy trực tiếp thông qua mã nguồn.
* Mã nguồn được chuyển đổi thành bytecode trung gian và sau đó thành ngôn ngữ mẹ đẻ của máy tính nội bộ bằng trình thông dịch Python.
* Mã được thực thi và đầu ra được trình bày.

### **Chương trình là gì?**

  
**Chương trình** là một tập hợp các hướng dẫn yêu cầu máy tính thực hiện một tác vụ cụ thể.

#### Ngôn ngữ lập trình là ngôn ngữ được sử dụng để tạo chương trình.

#### Trong khóa học này, chúng ta sẽ sử dụng Python để đưa ra hướng dẫn cho máy tính.

#### Điều này có nghĩa là tôi có thể làm cho máy tính của tôi làm bất cứ điều gì?

#### Vâng, đúng với giới hạn của nó.

#### Tất cả mọi thứ liên quan đến việc đưa ra hướng dẫn.

#### **Ví dụ:** Khi bạn nhấp vào nút **Phát** trên trình phát đa phương tiện và nhạc bắt đầu phát.

#### Có một chương trình hoạt động đằng sau hậu trường yêu cầu máy tính bật nhạc khi nhấp vào nút **Play**.

### **Viết chương trình đầu tiên của chúng tôi**

  
Không có tâm trạng để phá vỡ truyền thống và nghi lễ theo năm tháng.

#### Hãy bắt đầu với một chương trình **hello world**, thường là chương trình đầu tiên được học khi học một ngôn ngữ.

#### Chúng tôi muốn hiển thị "**Hello World**" trên màn hình đầu ra.

#### Nhưng, bằng cách nào?

#### Hãy xem nào.

#### Để yêu cầu máy tính viết trên màn hình, chúng tôi sử dụng câu lệnh **in**.

print()

#### Nó là một chức năng tích hợp trong python để hiển thị mọi thứ trên màn hình.

#### Một hàm tích hợp là một hàm được xác định trước và có thể được sử dụng trực tiếp.

#### Chính xác như bột mà mẹ bạn sử dụng để chuẩn bị bánh kếp.

#### Bất cứ điều gì chúng ta muốn hiển thị, cần phải được viết trong ngoặc đơn và được đặt trong dấu ngoặc kép " ".

#### Vì vậy, để hiển thị **Hello World**:

print(“Hello World”)

Chào thế giới

#### Vù vù!! Bạn vừa viết chương trình Python đầu tiên của mình.

### **Nhận xét trong Python**

#### Nhận xét là những đoạn mã bị trình thông dịch Python bỏ qua.

#### Nhận xét được viết để làm cho mã nguồn dễ hiểu hơn bởi người khác.

#### Python hỗ trợ nhận xét một dòng, có nghĩa là chúng chỉ có thể bao gồm một **dòng**.

#### Bất cứ điều gì được viết sau # được coi là một nhận xét dòng duy nhất trong Python.

# This is a comment in Python.

### **Tóm tắt**

  
Chúng tôi đã học được một chương trình là một tập hợp các hướng dẫn.

#### Hàm print() có thể được sử dụng để viết trong màn hình đầu ra.

#### Văn bản được in phải được đính kèm trong dấu ngoặc kép (" ")

#### print() là một hàm tích hợp sẵn.

#### # được sử dụng để viết chú thích bằng Python.

### **Biến là gì?**

#### Trong giá sách trên, chúng ta có thể thấy nhiều phần được dán nhãn chính xác.

#### Vì vậy, nếu chúng ta muốn đặt một cuốn sách toán học, chúng ta sẽ giữ nó trong phần được dán nhãn là Toán học, phải không?

#### Giá sách trên có thể được sử dụng để lưu trữ những cuốn sách mà chúng ta muốn.

#### Tương tự, trong lập trình, các biến hoạt động như các container để lưu trữ **các** giá trị.

#### Các giá trị này có thể là bất cứ thứ gì, từ số, đến văn bản hoặc toàn bộ hàm.

#### Và cũng giống như các nhãn trong trường hợp giá sách, các biến được xác định bằng **tên biến**.

#### Các giá trị được lưu trữ trong các biến có thể được sử dụng thêm trong các chương trình của chúng tôi.

#### Thế nào?

#### Chúng ta có thể truy xuất chúng bằng cách tham khảo các tên biến được sử dụng để xác định các biến này.

### 

### **Lưu trữ dữ liệu trong các biến**

#### Tên biến, giá trị và **toán tử gán** (=) là ba thứ cần thiết để lưu trữ các **giá trị trong biến**.

### **myVar = 60**

#### Ở đây, **myVar** là tên biến và nó được gán giá trị **60** bằng toán tử gán ('=').

#### Vì vậy, **myVar** là một biến có giá trị **60**.

#### Nếu bạn biết các ngôn ngữ khác, như C ++, bạn sẽ thấy rằng các biến không cần phải khai báo.

#### Chúng ta cũng có thể lưu trữ văn bản trong các biến.

#### Giá trị văn bản được gọi là **string** trong Lập trình.

#### Một chuỗi là một **string** các ký tự.

#### Các ký tự có thể là một chữ cái hoặc một số, thậm chí là một ký tự đặc biệt.

#### Bất cứ điều gì được viết giữa " "(dấu **ngoặc kép**) hoặc ' **'(dấu ngoặc đơn)** là một chuỗi trong Python.

myVar1 = “Hello”

#### Ở đây myVar1 là một **chuỗi** có giá trị "Xin chào".

#### "Hello 3", "456", "Hello", "45 Number" là những ví dụ về chuỗi trong Python.

#### Vì "456" được viết bên trong " " (dấu ngoặc kép), nó được coi là một **chuỗi** trong Python. Chúng ta cũng có thể lưu trữ số trong các biến.

#### Các số chứa dấu thập phân được gọi là số **dấu phẩy động** trong Lập trình.

#### Số dấu phẩy động đề cập đến một số bao gồm hai phần cách nhau bằng dấu thập phân.

#### Một số nguyên được gọi là **integer** trong Lập trình.

#### Hãy xem các số được lưu trữ trong Python như thế nào.

#### myVar = 2 # An integer

#### Ở đây, myVar có giá trị bằng 2 là một **số nguyên**.

#### myVar2 = 25.54 # A floating point number

#### Ở đây, myVar2 có giá trị 25,54 và do đó nó là một số dấu phẩy động.

#### Bất kể loại số nào, dù là dấu phẩy động hay số nguyên, Python xem chúng như một danh mục duy nhất - Số.

#### Python tự động xác định loại dựa trên các giá trị và chúng ta không cần chỉ định loại theo cách thủ công.

#### Đó là cách các loại biến khác nhau có thể được tạo ra trong Python.

### **Tóm tắt**

#### Trong Lập trình, các biến là các container có thể được sử dụng để lưu trữ các giá trị.

#### Giá trị có thể là số hoặc chuỗi.

#### Chuỗi được viết bên trong "" (dấu ngoặc kép) hoặc '' (dấu ngoặc đơn).

#### Các biến này được tham chiếu bằng cách sử dụng tên biến của chúng.

### **Cung cấp đầu vào cho chương trình**

#### Chúng ta biết cách lưu trữ các giá trị trong các biến và sử dụng chúng.

#### Nhưng, nếu bạn muốn lấy một số giá trị từ người dùng thì sao?

#### Bằng cách này, chương trình của bạn sẽ tương tác và linh hoạt.

#### Hãy xem làm thế nào chúng ta có thể lấy đầu vào từ người dùng trong Python.

### **Cung cấp đầu vào cho một chương trình**

#### **input()** được sử dụng trong Python để lấy đầu vào từ người dùng.

#### Cũng giống như print(), **input()** cũng là một hàm tích hợp.

myInputVar = input(‘Enter a number: ’)

#### myInputVar là biến sẽ lưu trữ giá trị được nhập bởi người dùng.

#### Chức năng đầu vào chấp nhận một chuỗi, sẽ được hiển thị cho người dùng khi được gọi.

#### Đầu ra của chương trình sẽ giống như

Nhập số:

#### Bên cạnh đó người dùng có thể nhập một giá trị.

Nhập số: 50

#### Nếu chúng ta thêm **print**(myInput Va) sau đoạn mã trước đó, chương trình sẽ trả về giá trị của **Input Var** trong cửa sổ đầu ra.

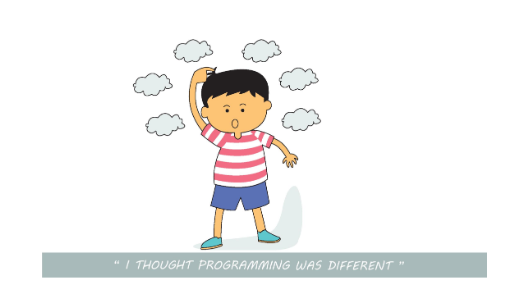
>print(myInputVar) => 50

### **Thêm hai số**

#### Bạn sẽ làm gì nếu ai đó yêu cầu bạn thêm hai số?

1. Bạn sẽ yêu cầu họ cung cấp hai số. Nói 5 và 6.
2. Thực hiện thao tác cộng - tức là tính 5 + 6, bằng 11.
3. Cho họ biết kết quả : 11

#### Bên phải?



#### Chúng tôi làm điều tương tự khi viết chương trình để thêm hai số.

#### Hãy viết một chương trình lấy hai số làm đầu vào từ người dùng và thêm chúng.

#### Chương trình cũng phải in kết quả lên màn hình.

#### Những gì chúng ta cần?

#### Một biến để giữ số đầu tiên, hãy gọi nó là **num1**.

#### Một biến khác để giữ số thứ hai, hãy gọi nó là **num2**.

#### Một biến thứ ba để lưu trữ **tổng** của num1 và num2, hãy để nó là tổng.

#### **Còn gì nữa**?

#### input() và () hàm.print

#### + toán tử để thực hiện phép cộng và = toán tử để lưu trữ kết quả.

#### Hãy bắt đầu.

num1 = int(input("Enter First number: ")) # taking input for first number

num2 = int(input("Enter Second number: ")) # taking input for second number

sum = num1 + num2 # adding num1 and num2 and storing them in sum

print(sum) # print sum to the screen

#### Lưu ý: Khi chúng tôi chấp nhận đầu vào từ người dùng bằng hàm input(), nó ở dạng 'chuỗi'. Vì vậy, ở đây trong ví dụ này, để thêm hai số, vì vậy chúng ta sẽ cần loại số.

#### Do đó, chúng ta sẽ cần phải chuyển nó sang dạng số. Chúng ta sử dụng int() để chuyển đổi thành số nguyên và float() để chuyển đổi thành float, v.v.

#### Bạn có thể đoán đầu ra của chương trình trên không?

### **Ra:**

#### Dòng đầu tiên yêu cầu người dùng nhập số đầu tiên.

#### Dòng thứ hai yêu cầu người dùng nhập số thứ hai.

#### Và dòng thứ ba in tổng của số thứ nhất và thứ hai.

#### Tương tự như vậy, chúng ta có thể viết các chương trình để trừ, nhân, chia hoặc bất kỳ hoạt động toán học nào khác mà chúng ta muốn làm.

#### Hãy nhớ rằng hàm đầu vào sẽ đợi người dùng cho đến khi nhập giá trị.

### **Tóm tắt**

#### Chúng ta đã học được input() được sử dụng để lấy đầu vào của người dùng.

#### Ngoài ra, chúng tôi đã học cách sử dụng các giá trị được lưu trữ trong các biến để thực hiện các thao tác và in kết quả lên màn hình.

### **Ra quyết định là gì?**

#### Chúng ta vô thức đưa ra quyết định trong mọi việc chúng ta làm.

#### Chẳng hạn

1. Có nên ăn không?
2. Nếu có, tôi nên ăn gì?
3. Pizza? Có.
4. Không đói, ngủ.

#### Bạn có thể đếm xem bạn vừa đưa ra bao nhiêu quyết định ở đó không?

### **Đưa ra quyết định bằng Python**

#### Giống như chúng ta đưa ra quyết định trong cuộc sống hàng ngày, chương trình của chúng ta cũng cần phải đưa ra quyết định.

#### Để đưa ra quyết định trong Python, chúng tôi có các tuyên bố **ra** **quyết định**.

#### Các tuyên bố ra quyết định do Python cung cấp là:

1. If
2. If-else
3. if-elif

#### Hãy xem cách chúng tôi có thể sử dụng những điều này để giúp các chương trình của chúng tôi đưa ra một số quyết định.

#### Câu lệnh **if** có thể được sử dụng để kiểm tra xem một điều kiện là đúng hay sai.

#### Nếu điều kiện được đánh giá là đúng, khối mã theo câu lệnh if được đánh giá.

#### Trong Python, để chỉ ra khối thụt lề mã được sử dụng.

#### Mỗi dòng phải được thụt lề bởi cùng một lượng trong một khối mã.

#### Một câu lệnh if điển hình trông giống như

if condition:

statements if true

#### Dấu **hai chấm** (:) được sử dụng trong Python để làm cho mã dễ đọc hơn và tách các phần khác nhau của biểu thức.

#### Hãy hiểu câu lệnh if sử dụng một ví dụ.

if 5 == 5:

print(“You successfully learned if statement.”)

#### Ở trên, nếu câu lệnh điều kiện 5==5 sẽ đánh giá là đúng. Và do đó, câu lệnh sau đây yêu cầu chương trình xuất ra "Bạn đã học thành công câu lệnh if." sẽ được in trên màn hình.

#### Toán tử == (bằng) được sử dụng để so sánh hai thực thể.

if 5 > 9:

print(“Oops! Not this time.”)

#### Trong trường hợp trên, nếu điều kiện câu lệnh được đánh giá là sai, do đó các câu lệnh sau đây nếu sẽ không bao giờ được tính toán.

#### Một câu lệnh **khác** thường được sử dụng với câu lệnh **if**.

#### Trong trường hợp khi điều kiện bên trong nếu đánh giá là sai, các câu lệnh theo câu lệnh khác được đánh giá.

#### Câu lệnh **if-else** trông giống như

if condition:

statements if true

else:

statements if false

#### else luôn được viết sau câu lệnh if.

#### Hãy hiểu nó bằng cách sử dụng một ví dụ.

#### Hãy xem xét đoạn trích dưới đây

if5 == 3:

print(“You successfully learned if statement.”)

else:

print(“Wow! You also learned about else statement.”)

#### Ở trên, nếu câu lệnh điều kiện 5==3 sẽ đánh giá là sai. Và do đó, tuyên bố sau đây nếu sẽ không được đánh giá nhưng tuyên bố sau đó sẽ.

#### Vì vậy, đầu ra sẽ là:

Wow! Bạn cũng đã học về tuyên bố khác.

#### Nếu chúng ta muốn kiểm tra nhiều điều kiện, chúng ta có thể sử dụng các câu lệnh **elif**.

if condition1:

statements

elif condition2:

statements

elif condition3:

statements

else:

statements

#### **Elif** sẽ luôn đến sau nếu tuyên bố, nó không thể tồn tại riêng lẻ.

#### Các câu lệnh ELIF sẽ được kiểm tra theo thứ tự mà chúng xuất hiện.

#### Hãy hiểu elif bằng cách sử dụng một ví dụ.

#### Chúng tôi có thể kiểm tra bao nhiêu điều kiện tùy thích.

if 5==4:

print(“An if statement. Oh!”)

elif 4==4:

print(“That’s something new.”)

elif 4>=9:

print(“Really?”)

else:

print(“Not this time”)

#### Trong trường hợp khối **if-elif**, các điều kiện sẽ được kiểm tra cho đến khi một điều kiện được tìm thấy là đúng.

#### Trong đoạn trích trên, điều kiện thứ hai sẽ đánh giá là đúng.

#### Như vậy, đầu ra sẽ là

Đó là một cái gì đó mới.

#### Nếu tất cả các điều kiện đánh giá là sai thì khối **khác** sẽ được thực thi.

### **Tóm tắt**

#### Giống như con người, máy tính cũng phải đưa ra một số quyết định.

#### Python cung cấp các câu lệnh ra quyết định if, if-else và if-elif-else.

#### Nếu có thể được sử dụng để kiểm tra một điều kiện duy nhất, else được sử dụng để cho máy tính biết phải làm gì trong trường hợp nếu câu lệnh được đánh giá là đúng.

#### Trong tình huống chúng ta cần kiểm tra nhiều giá trị, câu lệnh else if có thể được sử dụng.

### **Lặp lại một nhiệm vụ**

#### Điều gì sẽ xảy ra nếu ai đó yêu cầu bạn viết số từ một đến trăm

#### Hoặc nếu ai đó yêu cầu bạn viết tên của bạn hàng ngàn lần?

#### Một trong những giải pháp có thể là:

print(“1”)

print(“2”)

.

.

.

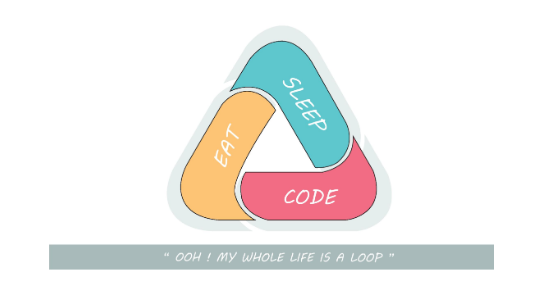
print(“99”)

print(“100”)

#### Nhưng viết cùng một tuyên bố 100 lần không phải là điều mà một người khôn ngoan sẽ làm.

#### Nó sẽ tiêu tốn nhiều thời gian hơn cũng như tăng số dòng trong code của bạn.

#### Python cung cấp cho chúng ta một cách hiệu quả để làm điều tương tự - Vòng lặp.



#### Vòng lặp cho phép chúng ta thực hiện một câu lệnh nhiều lần.

#### Có hai loại vòng lặp trong Python.

1. For loop
2. While loop

#### Hãy xem làm thế nào chúng ta có thể sử dụng chúng trong các chương trình của chúng tôi.

#### Vòng lặp A **for** lặp lại trên một chuỗi nhất định.

for variable in sequence:

statements

#### Trong đó biến đề cập đến một biến truy cập.

#### Biến này sẽ cập nhật mỗi khi vòng lặp được thực thi.

#### Và trình tự chứa dữ liệu mà vòng lặp cần được thực hiện.

#### Hãy xem một vòng lặp for trong hành động.

#### Hãy xem xét đoạn trích dưới đây

for num in range(1,101):

print(num)

#### Đoạn mã trên sẽ in các số từ 1 đến 100.

#### **range** là một hàm trong Python. range() lấy giá trị bắt đầu và giá trị kết thúc.

#### Nó tạo ra một danh sách các số giữa chúng bao gồm giá trị bắt đầu và không bao gồm giá trị kết thúc. Trong trường hợp của chúng tôi, từ 1 bao gồm đến 101 độc quyền.

#### Nếu chỉ có một giá trị được chuyển đến range(). Nó giả định rằng giá trị bắt đầu là 0 và tạo danh sách các số.

#### Phạm vi (100) và Phạm vi (0, 100) cho kết quả tương tự.

#### Một tham số thứ ba cũng có thể được chuyển đến hàm phạm vi, đại diện cho "bước".

#### Một bước là một số mà chỉ mục sẽ thay đổi mỗi khi vòng lặp thực thi.

#### Theo mặc định, bước là 1.

range(0,10,2)

#### Sẽ trở lại

[0, 2, 4, 6, 8]



#### Bạn có đoán được, chúng ta cần làm gì nếu chỉ muốn in các số lẻ từ 1 đến 10 không?

for num in range(1,10,2):

print(num)

#### Thao tác này sẽ trả về các số lẻ từ 1 đến 10, [1, 3, 5, 7, 9].

### **While loop**

#### Đã đến lúc khám phá **các** vòng lặp ngay bây giờ.

#### Một **vòng lặp thời gian** hoạt động dựa trên điều kiện.

while condition:

statements

#### Nó liên tục thực hiện các câu lệnh sau cho đến khi điều kiện được đánh giá là sai.

#### Trong đó điều kiện đại diện cho **điều kiện** cần được kiểm tra trước mỗi lần vòng lặp được thực hiện.

#### Và các tuyên bố đề cập đến nhiệm vụ mà chúng tôi muốn lặp lại.

#### Hãy xem trong khi vòng lặp trong hành động.

num = 0

while (num < 10):

print(num)

num = num + 1 #update the value of num by 1

#### Trong đoạn mã trên, một biến num được cho một giá trị 0.

#### Trong vòng lặp while, một điều kiện được thông qua có nội dung "Lặp lại những điều sau cho đến khi num nhỏ hơn 10".

#### Và sau khi tuyên bố in, chúng tôi yêu cầu chương trình cập nhật giá trị của num bằng 1. Điều này hoàn toàn tương tự như bước trong phạm vi.

#### Đoạn mã trên sẽ in các số từ 0 đến 9 trên màn hình.

### **Tóm tắt**

#### Vòng lặp được sử dụng để thực thi các câu lệnh nhiều lần.

#### Trong Python, **for** và **while** các vòng lặp được cung cấp.

#### Vòng lặp **a for** nhận biến và danh sách các số, có thể được tạo ví dụ bằng cách sử dụng range().

#### Một thời gian mất một điều kiện và liên tục thực hiện các câu lệnh cho đến khi điều kiện được tìm thấy là đúng.

### **Chức năng là gì?**

#### Điều gì sẽ xảy ra nếu bạn có thể tránh gõ đi gõ lại cùng một đoạn mã nếu bạn cần nó trong nhiều phần của chương trình?

#### Nó sẽ không tuyệt vời?

#### Chức năng là giải pháp cho vấn đề trên.

#### Chức năng là một tập hợp các hướng dẫn thực hiện một tác vụ cụ thể và có thể được sử dụng lại nhiều lần.

#### Nhiệm vụ cụ thể có thể là thêm hai số hoặc tìm số lớn nhất trong danh sách, để đưa ra một ví dụ.

#### Tại sao chúng ta sử dụng các hàm có thể được hiểu từ đoạn mã dưới đây.

sum1 = num1 + num2

sum2 = num3 + num4

sum3 = num5 + num6

#### Nếu chúng ta muốn thực hiện phép cộng các biến được liệt kê ở trên.

#### Chúng ta cần viết những tuyên bố tương tự 3 lần.

#### Các hàm cho phép chúng ta thực hiện thao tác cộng tương tự mà không cần viết cùng một thao tác thêm nhiều lần.

#### Hãy cùng khám phá cách thực hiện.

### **Chức năng viết**

#### Các hàm trong Python được viết như mô tả bên dưới

def functionName (parameters):

statements

return something

#### **def** được sử dụng để khai báo một hàm.

#### **function Name** là tên của hàm.

#### Một hàm có thể nhận các **tham số**, hoặc có thể không. Các tham số bao gồm các biến được hàm sử dụng, giống như hai số được tính tổng.

#### **Các câu lệnh** là các hoạt động mà chúng ta cần thực hiện.

#### Và, câu lệnh trả về đánh dấu sự kết thúc của hàm và có thể được sử dụng để **trả về** đầu ra. Bạn có thể tránh đặt nó nếu bạn không phải trả lại bất kỳ giá trị nào.

#### \* Chúng tôi sẽ tìm hiểu thêm về các thông số trong tương lai.

#### Hãy bắt đầu bằng cách viết một hàm đơn giản.

def helloWorld():

print(“Hello World”)

#### Vậy là xong, chúng ta đã viết hàm đầu tiên của mình bằng Python.

#### Chức năng này không nhận thông số và in "Hello World" mỗi khi nó được gọi.

#### Vì vậy, bây giờ nếu tôi muốn in hello world nhiều lần, tôi không cần phải viết print ("Hello World") nhiều lần. Tôi chỉ có thể gọi hàm trên.

#### Chúng ta sẽ sớm tìm hiểu cách gọi một hàm.

#### Hãy viết một hàm lấy tham số và trả về một cái gì đó.

def addNumbers(num1, num2):

sum = num1 + num2

return sum

#### Hàm trên lần lượt lấy hai tham số num1 và num2, lưu trữ kết quả cộng trong tổng biến và trả về nó.

### **Gọi một hàm**

#### Đủ các chức năng xác định. Hãy xem làm thế nào chúng ta có thể sử dụng chúng.

#### Để sử dụng các hàm mà chúng ta đã định nghĩa, chúng ta cần thực hiện một cuộc gọi đến hàm.

#### Gọi một hàm dùng cú pháp sau đây:

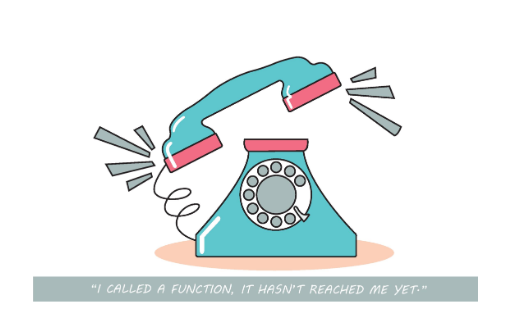
functionName(parameters)

#### Đó là nó. Để gọi bất kỳ hàm nào, chúng ta chỉ cần chỉ định tên và tham số của nó trong ngoặc, nếu có.



#### Bây giờ, bạn có thể đoán những gì chúng tôi đã làm kể từ đầu thời gian khi chúng tôi viết in ("Một số thứ điên rồ đang diễn ra ở đây")?

#### Chúng tôi thực sự đang thực hiện một cuộc gọi đến chức năng in và chuyển một tham số mà chúng tôi muốn in ra màn hình.



#### print() là một hàm được tích hợp sẵn, có thể có hàng trăm dòng được viết bằng Python để làm cho nó hoạt động.

#### Nếu không có chức năng, chúng tôi có thể đã viết lại tất cả những dòng đó bất cứ khi nào chúng tôi cần viết một cái gì đó để sàng lọc.

#### Hãy xem việc gọi một hàm thông qua một vài ví dụ.

def helloWorld(): # Define the helloWorld function.

print(“Hello World”)

helloWorld() # Call to helloWorld function.

#### Ở trên sẽ in **Hello World** trên màn hình đầu ra.

#### Chúng ta có thể gọi hàm bao nhiêu lần tùy thích và nó sẽ in **Hello World** cùng số lần.

#### Hãy cùng khám phá cách chúng ta có thể truyền các tham số và thực hiện các thao tác trên các tham số đó.

def addNumbers(num1, num2):

sum = num1 + num2

print(sum)

return

add Numbers(2,3)

add Numbers(4,5)

#### Chúng ta đang gọi hàm add Numbers ở đây và truyền hai số để thực hiện thao tác cộng. Chúng tôi đã gọi hàm hai lần.

#### Đầu ra của đoạn trích trên sẽ là

5 9 >

### **Tóm tắt**

#### Chức năng là các khối mã có thể tái sử dụng được viết để thực hiện các tác vụ cụ thể.

#### Các hàm trong Python được khai báo bằng từ khóa **def**.

#### **def** được theo sau bởi tên hàm theo sau là các tham số của hàm. Phần thân hàm chứa các câu lệnh sẽ được thực thi.

#### Các hàm có thể được gọi bằng tên hàm của chúng, theo sau là dấu ngoặc và tham số nếu có.

### **Thao tác trên số và chuỗi**

#### Python cung cấp nhiều hàm tích hợp có thể được sử dụng để thực hiện các hoạt động phức tạp.

#### Trong phần này, chúng ta sẽ khám phá một số trong số chúng để thao tác với các số và chuỗi của chúng ta.

#### Hãy bắt đầu.

### **Hàm số**

#### Hàm đầu tiên trong danh sách là hàm **pow().**

#### Nó lấy hai số và trả về kết quả của số đầu tiên được nâng lên lũy thừa của số thứ hai.

#### Nhưng để sử dụng tất cả các hàm mà chúng ta sẽ thảo luận, chúng ta cần nhập môđun **toán học**.

#### **Mô-đun** là một tệp Python bên ngoài, chứa các hàm được sử dụng bởi các chương trình Python khác.

#### Để nhập một câu lệnh **nhập** mô-đun được sử dụng.

import moduleName

#### Trong trường hợp của chúng tôi, chúng tôi cần môđun toán học. Vì vậy, chúng tôi sẽ viết

import math

#### Để truy cập bất kỳ hàm nào của mô-đun toán học, chúng ta viết **math.function Name()**

#### Trong trường hợp pow.

import math

math.pow(4,3)

#### Điều này sẽ trả lại 4 nâng lên lũy thừa của 3 là 64.

#### Chúng ta sẽ làm theo cú pháp tương tự để sử dụng các hàm toán học khác.

### **Chức năng floor và ceil**

#### Chức năng sàn và ceil là các chức năng được sử dụng rộng rãi khác.

#### **math.floor()** lấy một số làm tham số và trả về số nguyên lớn nhất bằng hoặc nhỏ hơn số được truyền làm tham số.

import math

a = math.floor(4.3)

b = math.floor(10.9)

print (a)

print (b)

#### Đầu ra của đoạn trích trên sẽ là

4 10

#### math.ceil() lấy một số làm tham số và trả về số nguyên nhỏ nhất bằng hoặc lớn hơn số được truyền làm tham số.

import math

a = math.ceil(4.3)

b = math.ceil(10.9)

print (a)

print (b)

#### Đầu ra của đoạn trích trên sẽ là

5 11

### **Hàm ABS()**

#### Hàm abs() nhận một tham số và trả về giá trị tuyệt đối của giá trị được truyền.

#### ABS() không phải là một phần của mô-đun toán học. Nó là một chức năng tích hợp.

#### Giá trị tuyệt đối là độ lớn của số không phân biệt dấu của nó, tức là +ving hoặc -ved.

a = abs(10)

b = abs(-10)

print (a)

print (b)

#### Đầu ra của đoạn trích trên sẽ là

10 10

### **Tìm nhật ký số trong Python**

#### Có, môđun toán học cũng cung cấp một hàm để tìm logarit của số.

#### **math.log()** có thể lấy một hoặc hai tham số.

#### Khi một tham số được truyền, nó trả về **logarit tự nhiên** của số đó.

#### Khi hai tham số được truyền, nó trả về logarit của số đầu tiên đến cơ số của số thứ hai.

import math

a = math.log(10)

b = math.log(10,2)

print (a)

print (b)

#### Đầu ra của đoạn trích trên sẽ là

2.30258509299 3.32192809489

### **Tìm căn bậc hai của số trong Python**

#### Căn bậc hai của một số có thể được tìm thấy bằng cách sử dụng hàm **math.sqrt().**

#### math.sqrt() nhận một tham số duy nhất và trả về căn bậc hai của số.

import math

a = math.sqrt(9)

b = math.sqrt(16)

print (a)

print (b)

#### Đầu ra của đoạn trích trên sẽ là

3.0 4.0

#### Ngoài các hàm này, môđun toán học bao gồm nhiều hàm khác cũng bao gồm các hàm lượng giác và hyperbolic.

#### Chúng tôi sẽ không khám phá những điều đó trong khóa học này, nhưng hãy kiểm tra chúng!

#### Đã đến lúc xem xét lại các chuỗi

### **Chuỗi là gì?**

#### Dữ liệu văn bản trong ngôn ngữ lập trình được biểu diễn bằng chuỗi.

#### Một chuỗi là một chuỗi các ký tự. Nó cũng có thể chứa khoảng trắng và số.

#### Có ba cách để viết một chuỗi bằng Python.

num1 = 'this is a string.'

Using single quotes ''

num2 = "this is also a string."

Using double quotes ""

#### Python xử lý dấu ngoặc kép và dấu ngoặc kép tương tự nhau, nhưng khi đặt một chuỗi, cần sử dụng cùng một loại dấu ngoặc kép ("điều này là sai").



num3 = '''this looks somewhat different

It is really different you can see multiple lines in this string

Isn't it awesome'''

#### Một chuỗi được đặt trong dấu ngoặc kép ba ''' '' có thể kéo dài đến nhiều dòng và tất cả khoảng trắng sẽ được bao gồm trong chuỗi. """ "" cũng có thể được sử dụng thay cho ''' '''.

#### Chúng ta hãy xem một số hàm tích hợp mà chúng ta có thể sử dụng để thao tác với các chuỗi này.

### **Hàm CAPITALIZE()**

#### Nó trả về chuỗi trong trường hợp Câu, tức là với chữ cái đầu tiên được viết hoa và phần còn lại nhỏ.

#### Để sử dụng các hàm chuỗi, chúng ta viết

str.functionName()

#### Trong đó **str** là chuỗi mà chúng ta muốn thao tác.



#### Vì vậy, trong trường hợp viết hoa()

str = "here I Am."

str.capitalize()

#### Điều này sẽ trở lại **Here i am**. Lưu ý rằng giá trị của str vẫn giữ nguyên, capitalize() trả về một chuỗi mới!

### **Hàm COUNT()**

#### Hàm **str.count()** trả về số lần xuất hiện của chuỗi con được chỉ định làm tham số.

str = "here I Am."

str.count(“e”)

#### str.count sẽ trả về 2.

#### B e nguyên nhân "**e**" có hai lần xuất hiện trong str.

#### Tương tự

str = "here I Am."

str.count(‘Am’) #will return 1.

### **Hàm find()**



#### Một **str**. **Hàm** find() trả về vị trí của lần xuất hiện đầu tiên của chuỗi con được truyền trong find() dưới dạng tham số nếu nó tồn tại trong str.

#### index là thuật ngữ dùng để chỉ các vị trí trong lập trình.

#### Các chỉ số này được thể hiện bằng số nguyên, tương tự như số nhà của chúng tôi.

#### Trong trường hợp này, đây là những con số để biểu thị vị trí bộ nhớ.

#### Ký tự đầu tiên sẽ luôn ở vị trí 0., tức là chỉ số bắt đầu từ 0.

str = "here I Am."

a = str.find("h")

print (a)

#### Đầu ra của đoạn trích trên sẽ là

0

#### Bởi vì h xảy ra ở chỉ số 0. Hãy nhớ rằng, các chỉ số bắt đầu từ 0, vì vậy điều đó có nghĩa là vị trí đầu tiên!

#### Tương tự, str.find('I') sẽ trả về 5.

#### Ngoài ra, str.find('d') sẽ trả về -1 vì d không được tìm thấy trong chuỗi.

### **Hàm join()**

#### Hàm **str.join()** trả về chuỗi được ghép nối của chuỗi chuỗi chuỗi được truyền đến nó.

#### Trong trường hợp này, **str** chứa dấu phân cách được sử dụng trong khi nối các chuỗi đó.

str = " "

iter = ("I","am","awesome.")

a = str.join(iter)

print(a)

#### Đầu ra của đoạn trích trên sẽ là

Tôi thật tuyệt vời.

#### Tương tự

str = "-"

iter = ("I","am","awesome.")

a = str.join(iter)

print(a)

#### Đầu ra của đoạn trích trên sẽ là

I-am-awesome.

### **Danh sách là gì?**

#### Danh sách là một cấu trúc dữ liệu trong Python được sử dụng để chứa nhiều giá trị, giống như những gì bạn làm với danh **sách** mua sắm

#### Các giá trị này có thể có hoặc không cùng loại.

#### **Cấu trúc** dữ liệu là một định dạng để lưu trữ dữ liệu.

### **Tạo danh sách**

#### Danh sách chứa các mục được phân tách bằng dấu phẩy.

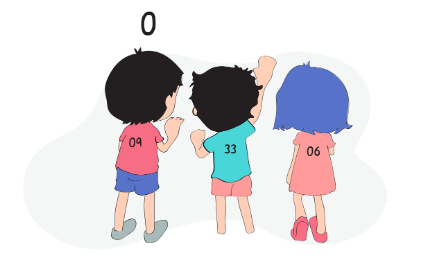
#### Các mục này được đặt trong dấu ngoặc vuông.

#### Một biến có thể được sử dụng để lưu trữ một **danh sách**.

list1 = [2, 4, 5, 6, 7, 2]

#### Một lời nhắc nhở thân thiện, một **danh sách** có thể lưu trữ các giá trị của các loại khác nhau như chuỗi, số, v.v.

list2 = [4, ”hi”, 6, ”Me”, 78]



#### Các mục trong danh sách được xác định bằng cách sử dụng vị trí của chúng.

#### Những vị trí này được gọi là **chỉ số**.

#### Chỉ số bắt đầu từ 0.

list1 = [2, 4, 5, 6, 7, 2]

#### Do đó, trong danh list1 được định nghĩa ở trên, 2 ở chỉ số 0, 4 ở chỉ số 1, 5 ở chỉ số 2, v.v.

### **Truy cập dữ liệu từ Danh sách**

#### Các mục trong danh sách được truy cập bằng cách sử dụng số chỉ mục của chúng.

#### Để truy cập một mục trong danh sách, chúng tôi viết tên của danh sách, theo sau là chỉ mục của mục chúng tôi muốn truy cập bên trong dấu ngoặc vuông.

list1 = [2, 4, 5, 6, 7, 2]

#### Nếu chúng ta muốn truy cập giá trị 4, chúng ta phải viết

list1[1]

#### Tên của danh sách theo sau là số chỉ mục là 1 bên trong dấu ngoặc vuông.

#### Viết print(list1[1]) sẽ hiển thị **4**. Ở đây bạn có thể thấy rằng bạn có thể truyền các phần tử phức tạp vào hàm print()

#### Chúng tôi cũng có thể truy cập nhiều mục tiếp theo trong một danh sách.

#### Để làm như vậy, chúng ta cần chỉ định chỉ mục bắt đầu và chỉ mục kết thúc, được phân tách bằng dấu hai chấm.

list1 = [2, 4, 5, 6, 7, 2]

#### Vì vậy, nếu chúng ta muốn truy cập các giá trị 4, 5, 6, chúng ta có thể viết

list1 = [2, 4, 5, 6, 7, 2]

a = list1[1:4]

print(a)

#### Và đầu ra của đoạn trích trên sẽ là

[4, 5, 6]

#### Để truy cập tất cả các mục, chúng ta có thể viết tên của danh sách hoặc lập chỉ mục tất cả các phần tử một cách rõ ràng

list1 = [2, 4, 5, 6, 7, 2]

list1[:]

#### Điều này sẽ trở lại.

[2, 4, 5, 6, 7, 2]

#### Tương tự, để truy cập các mục bắt đầu từ index 0, chúng ta có thể viết

list1[:5]

#### Khi không có chỉ mục nào được chỉ định trước dấu hai chấm, việc lập chỉ mục bắt đầu từ 0.To truy cập các phần tử từ chỉ mục 2 cho đến cuối chúng ta có thể viết

list1[2:]

#### Khi không có chỉ mục nào được chỉ định sau dấu hai chấm, quá trình lập chỉ mục kết thúc ở phần tử cuối cùng.

### **Cập nhật dữ liệu trong Danh sách**

#### Cập nhật **danh sách** là một miếng bánh.

#### Chúng tôi biết cách truy cập các mục riêng lẻ, phải không?

#### Bây giờ chúng ta chỉ cần cập nhật giá trị của các mục đó.

#### Để làm như vậy, trước tiên chúng ta sẽ truy cập mục và gán một giá trị mới cho nó.

#### Ví dụ, hãy xem xét một danh sách

list2 = [4, 55, 6, 7, 8, 9, 90]

#### Để cập nhật giá trị của mục tại chỉ mục 4 (có giá trị 8), chúng tôi viết

list2[4] = 88

#### Sau đó, danh sách cập nhật trông giống như [4, 55, 6, 7, 88, 9, 90]

#### Chúng tôi cũng có thể thêm một mục vào cuối thay vì thay thế những mục hiện tại. Hãy xem làm thế nào.

#### Để thêm một mục vào cuối danh sách, chúng ta sử dụng hàm append().

list2.append(150)

#### Sẽ thêm 150 vào cuối danh sách và danh sách cập nhật sẽ giống như

[4, 55, 6, 7, 88, 9, 90, 150]

#### Tương tự, liên tục sử dụng append(), bạn có thể thêm bao nhiêu phần tử bạn muốn trong danh sách (có thể với một vòng lặp?)

### **Xóa dữ liệu trong Danh sách**

#### Giống như chúng ta có thể thêm các mục vào danh sách, chúng ta cũng có thể xóa.

#### Để del ete các mục từ danh sách, chúng tôi sử dụng câu lệnh **del** theo sau là tên của danh sách và chỉ mục trong ngoặc.

#### Hãy xem xét, chúng tôi muốn xóa mục thứ 3 trong list1 = [4, 5, 6, 2 ,1]

#### Để làm như vậy, chúng tôi viết

del list1[3]

#### Thao tác này sẽ xóa mục ở chỉ mục 3 và trả về [4, 5, 6, 1]

#### Chúng ta có thể sử dụng hàm remove() để có được kết quả tương tự.

#### Trong remove(), chúng ta cần chỉ định phần tử mà chúng ta muốn xóa.

list1 = [4, 5, 6, 2 ,1]

list1.remove(5)

print(list1)

#### Đoạn mã trên sẽ xóa 5 khỏi danh sách và sẽ hiển thị [4, 6, 2, 1]

### **Tóm tắt**

#### Chúng tôi đã học được một **danh sách** là một cấu trúc dữ liệu có thể chứa nhiều giá trị khác nhau hoặc cùng loại.

#### Chúng ta có thể thêm các mục vào danh sách bằng hàm.append()

#### Ngoài ra, chúng tôi có thể cập nhật bất kỳ giá trị nào bằng cách sử dụng ký hiệu dấu ngoặc và chỉ mục của mục theo sau là một giá trị mới.

#### Để delete, bất kỳ câu lệnh **del** mục nào cũng có thể được sử dụng, tương tự, hàm cũng được sử dụng để xóa các mục khỏi danh sách.remove()

### **Bộ dữ liệu là gì?**

#### Bộ dữ liệu là một cấu trúc **dữ** liệu giống như danh sách được sử dụng để chứa nhiều giá trị.

#### Các giá trị này có thể có hoặc không cùng loại.

#### Nhưng nếu nó hoàn toàn giống với một danh sách, thì chúng ta sẽ không cần nó. Bên phải?

### **Tạo một bộ dữ liệu**

#### Bộ dữ liệu là một chuỗi các mục được phân tách bằng dấu phẩy.

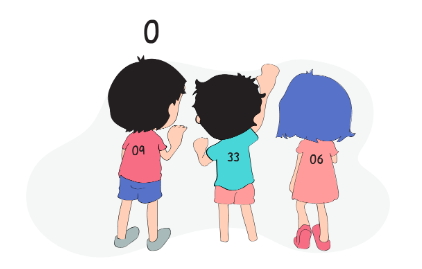
#### Các mục này được đặt trong **ngoặc đơn**.

#### Một biến có thể được sử dụng để lưu trữ một tuple.

tuple1 = (2, 4, 5, 6, 7, 2)

#### Một bộ giống như một danh sách có thể lưu trữ các giá trị của các loại khác nhau

tuple2 = (4, ”hi”, 6, ”Me”, 78)



#### Cũng giống như danh sách, các mục trong bộ dữ liệu cũng được xác định bằng các chỉ mục.

#### Trong tuples cũng chỉ số bắt đầu từ 0.

tuple1 = (2, 4, 5, 6, 7, 2)

#### Do đó, trong tuple1 được định nghĩa ở trên, 2 ở chỉ số 0, 4 ở chỉ số 1, 5 ở chỉ số 2, v.v.

### **Truy cập dữ liệu từ Tuple**

#### Các mục trong bộ dữ liệu được truy cập bằng cách sử dụng số chỉ mục của chúng.

#### Để truy cập một mục trong bộ dữ liệu, chúng tôi viết tên của bộ theo sau là chỉ mục của mục chúng tôi muốn truy cập bên trong dấu ngoặc vuông, giống như danh sách.

tuple1 = (2, 4, 5, 6, 7, 2)

#### Vì vậy, nếu chúng ta muốn truy cập giá trị 4, chúng ta phải viết

tuple1[1]

#### Tên của bộ theo sau là số chỉ mục là 1 bên trong dấu ngoặc vuông.

#### Viết in (tuple1 [1]) sẽ hiển thị 4.

#### Chúng tôi cũng có thể truy cập nhiều mục tiếp theo trong một bộ giống như cách chúng tôi làm trong danh sách.

#### Để làm như vậy, chúng ta cần sử dụng chỉ mục bắt đầu theo sau là dấu hai chấm theo sau là chỉ mục của mục mà chúng ta muốn truy cập các mục.

tuple1 = (2, 4, 5, 6, 7, 2)

#### Vì vậy, nếu chúng ta muốn truy cập các giá trị 4, 5, 6, chúng ta phải viết

tuple1[1:4]

#### Viết bản in (tuple1 [1: 4]) sẽ hiển thị (4, 5, 6).

#### Để truy cập tất cả các mục chúng ta có thể viết

tuple1

#### Điều này sẽ trở lại (2, 4, 5, 6, 7, 2)

#### Tương tự, để truy cập các mục bắt đầu từ index 0, chúng ta có thể viết

tuple1[:5]

#### Thao tác này sẽ trả về các mục bắt đầu từ index 0 đến index 5.

#### Và viết tuple1[2:] sẽ trả về tất cả các mục bắt đầu từ index 2 cho đến cuối tuple.

### **Cập nhật, xóa**

#### Không giống như danh sách, bộ dữ liệu là bất biến.

#### **Bất biến** đề cập đến thực tế là các mục trong bộ dữ liệu không thể được cập nhật hoặc xóa.

#### Vì vậy, không có cập nhật hoặc xóa các mục trong tuples.

#### Nhưng chúng ta có thể xóa toàn bộ bộ tuple. Hãy xem làm thế nào.

#### Để delete tuple, chúng tôi sử dụng câu lệnh **del**.

#### del theo sau là tên của tuple.

#### Vì vậy, để xóa tuple1 chúng ta cần viết

del tuple1

### **Tóm tắt**

#### Chúng tôi đã học được một tuple là một cấu trúc dữ liệu có thể chứa nhiều giá trị khác nhau hoặc cùng loại.

#### Chúng tôi không thể cập nhật bất kỳ mục nào trong một bộ dữ liệu.

#### Nhưng chúng ta có thể xóa toàn bộ bộ dữ liệu bằng cách sử dụng câu lệnh del.

#### Tại sao sử dụng tuples sau đó? Vâng, có những trường hợp một tuple được ưa thích hơn danh sách.

### **Dictionary là gì?**

#### Giống như một cuốn từ điển có các từ và ý nghĩa của chúng.

#### **Từ điển** là một cấu trúc dữ liệu chứa dữ liệu dưới dạng các cặp **khóa** và **giá trị**.

#### Một cặp khóa và giá trị tạo thành một mục trong từ điển. Một phím thường là một chuỗi

### **Tạo từ điển**

#### Các mục trong từ điển được phân tách bằng dấu phẩy. Lưu ý rằng mục cuối cùng không có dấu phẩy ở cuối.

#### Khóa và giá trị được phân tách bằng dấu hai chấm (:).

#### Các mục trong từ điển được đặt trong dấu ngoặc nhọn.

dict1 = {

‘name’ : ‘xyz’,

‘age’ : 25,

‘hobby’ : ‘Dancing’

}

#### dict1 là một từ điển chứa ba mục.

#### Đầu tiên là 'name' với giá trị 'xyz'.

#### Thứ hai là 'tuổi' với giá trị 25.

#### Và, 'sở thích' với giá trị 'nhảy múa'.

#### Hãy xem làm thế nào chúng ta có thể truy cập các mục trong từ điển.

### **Accessing data from dictionary**

#### Các mục trong từ điển được truy cập bằng cách sử dụng các khóa của chúng.

#### Để truy cập một mục trong từ điển, chúng tôi viết tên của từ điển theo sau là khóa của mục chúng tôi muốn truy cập bên trong dấu ngoặc vuông.

dict1 = {

‘name’ : ‘xyz’,

‘age’ : 25,

‘hobby’ : ‘Dancing’

}

#### Vì vậy, nếu chúng ta muốn tiếp cận giá trị của 'tuổi', chúng ta phải viết

dict1[‘age’]

#### Viết bản in (dict1 ['tuổi']) sẽ hiển thị 25.

#### Tương tự, viết print(dict1['name']) sẽ hiển thị 'xyz'.

#### Và viết print (dict1) sẽ hiển thị tất cả các mục

{'hobby': 'Dancing', 'age': 25, 'name': 'xyz'}

#### Lưu ý rằng các mục không theo thứ tự, điều này bởi vì từ điển được coi là cấu trúc dữ liệu không có thứ tự.

#### Nếu chúng tôi cố gắng truy cập một khóa không có trong từ điển, chúng tôi sẽ gặp lỗi.

### **Cập nhật dữ liệu trong từ điển**

#### Trong từ điển, chúng ta có thể thêm cặp khóa-giá trị mới hoặc cập nhật cặp hiện có.

#### Để cập nhật một mục nhập hiện có, trước tiên chúng ta cần truy cập vào mục và sau đó gán một giá trị mới cho nó.

#### Chẳng hạn

dict1 = {

‘name’ : ‘xyz’,

‘age’ : 25,

‘hobby’ : ‘Dancing’

}

#### Để cập nhật giá trị của '**tên**', chúng tôi viết,

dict1[‘name’] = ‘abc’

#### Đó là nó. Tên sẽ được cập nhật để chứa giá trị 'abc'.

#### Để thêm một cặp khóa-giá trị mới vào từ điển, chúng ta có thể sử dụng một khóa không có trong từ điển và gán một giá trị cho nó.

#### Ví dụ: viết dict1['profession'] = 'pilot' sẽ thêm một khóa mới 'profession' với giá trị 'pilot' vào dict1.

#### Và bây giờ, dict1 cập nhật sẽ trông như thế nào.

dict1 = {

‘name’ : ‘xyz’,

‘age’ : 25,

‘hobby’ : ‘Dancing’,

‘Profession’ : ‘pilot’

}

#### Hãy xem làm thế nào chúng ta có thể xóa các mục trong từ điển.

### **Xóa dữ liệu trong từ điển** Trong từ điển, chúng ta có thể xóa một mục riêng lẻ hoặc toàn bộ từ điển.

#### Để xóa một mục riêng lẻ, **câu lệnh del** được sử dụng, theo sau là tên của từ điển và khóa bên trong dấu ngoặc vuông.

### **del dict1[‘name’]**

#### Câu lệnh sẽ xóa mục có khóa 'tên' khỏi từ điển.

### 

#### Để loại bỏ toàn bộ câu lệnh **del** từ điển được sử dụng theo sau là tên của từ điển.

del dict1

#### Sẽ xóa toàn bộ từ điển.

#### Thay vì xóa toàn bộ từ điển, chúng ta cũng có thể chỉ cần xóa tất cả các mục và làm trống từ điển.

#### Để làm như vậy, chúng ta cần sử dụng hàm.clear()

#### Viết dict1.clear() sẽ xóa tất cả các mục khỏi dict từ điển 1.

#### Nếu chúng ta viết print(dict1) sau câu lệnh rõ ràng. Chúng ta sẽ lấy {} làm đầu ra, cho biết một từ điển không có mục.

### **Tóm tắt**

#### Chúng tôi đã học được từ **điển** là một cấu trúc dữ liệu chứa các cặp **khóa** và **giá trị** được phân tách bằng dấu phẩy và được đặt trong dấu ngoặc nhọn.

#### Để truy cập bất kỳ mục nào trong từ điển, chúng ta có thể sử dụng tên từ điển theo sau là tên khóa bên trong dấu ngoặc vuông.

#### Để cập nhật bất kỳ mục nào, trước tiên chúng ta truy cập vào mục đó và gán cho nó giá trị chúng ta muốn.

#### Để delete một mục, chúng ta có thể sử dụng câu lệnh **del** theo sau là tên từ điển và khóa trong ngoặc.

#### Để loại bỏ tất cả các mục chúng ta có thể sử dụng hàm **clear().**

#### Và, để delete toàn bộ từ điển, chúng ta có thể sử dụng **del** theo sau là tên từ điển.