**Python basic**

* List[] and Tuples(): both contains items but tuples cannot be **UPDATED (read-only).**

***Phân biệt Dictionary{}(dict), List[] và Tuple() trong Python***

**Dictionary:**

* Là một cấu trúc dữ liệu lưu trữ dữ liệu dưới dạng cặp **khóa-giá trị**. “key”:value
* Khóa là duy nhất và không được phép trùng lặp.
* Giá trị có thể là bất kỳ kiểu dữ liệu nào trong Python.
* Truy cập dữ liệu trong dictionary bằng cách sử dụng khóa.
* Ví dụ:

Python

my\_dict = {"name": "Bard", "age": 2, "language": "Python"}

print(my\_dict["name"]) # Output: Bard

Để lấy giá trị keys thì dùng lệnh : nameOfDict.keys()

Để lấy giá trị values thì dùng lệnh : nameOfDict.values()

**List:**

* Là một cấu trúc dữ liệu lưu trữ dữ liệu dưới dạng **danh sách có thứ tự**.
* Các phần tử trong list có thể là bất kỳ kiểu dữ liệu nào trong Python.
* Truy cập dữ liệu trong list bằng cách sử dụng chỉ mục.
* Ví dụ:

Python

my\_list = ["a", 1, True, [1, 2, 3]]

print(my\_list[2]) # Output: True

**Tuple:**

* Giống như list, tuple là một cấu trúc dữ liệu lưu trữ dữ liệu dưới dạng **danh sách có thứ tự**.
* Tuy nhiên, tuple **không thể thay đổi** sau khi được tạo.
* Ví dụ:

Python

my\_tuple = ("a", 1, True, [1, 2, 3])

# my\_tuple[2] = False # TypeError: 'tuple' object does not support item assignment

**Bảng so sánh:**

|  |  |  |  |
| --- | --- | --- | --- |
| Tính năng | Dictionary | List | Tuple |
| Loại dữ liệu | Cặp khóa-giá trị | Danh sách có thứ tự | Danh sách có thứ tự |
| Thay đổi | Có thể thay đổi | Có thể thay đổi | Không thể thay đổi |
| Truy cập | Sử dụng khóa | Sử dụng chỉ mục | Sử dụng chỉ mục |
| Tốc độ truy cập | Nhanh | Nhanh | Nhanh hơn list |
| Sử dụng | Lưu trữ dữ liệu có liên quan | Lưu trữ dữ liệu có thứ tự | Lưu trữ dữ liệu không thay đổi |

**Ví dụ sử dụng:**

* **Dictionary:** Thích hợp để lưu trữ dữ liệu có liên quan, ví dụ như thông tin cá nhân, thông tin sản phẩm, v.v.
* **List:** Thích hợp để lưu trữ dữ liệu có thứ tự, ví dụ như danh sách tên, danh sách số, v.v.
* **Tuple:** Thích hợp để lưu trữ dữ liệu không thay đổi, ví dụ như tọa độ, thông tin cấu hình, v.v.

**Kết luận:**

Dictionary, list và tuple là ba cấu trúc dữ liệu cơ bản trong Python. Việc lựa chọn sử dụng loại nào phụ thuộc vào mục đích sử dụng và đặc điểm của dữ liệu.

* Python phân biệt thụt đầu dòng cho 1 nhóm lệnh

print("Could not find true expression")

print("Good bye!")

#2 lệnh trên đi chung với nhau

print("trueGood bye!")

***LOOP***

**Vòng lặp while (while loop):**

* **Chức năng:** Lặp đi lặp lại một khối mã lệnh **miễn là một điều kiện nhất định vẫn đúng**.
* **Cấu trúc:**

Python

while điều\_kiện:

khối\_lệnh

example:

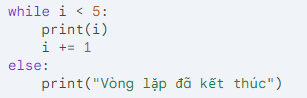
i = 1

while i <= 5:

print(i)

i += 1 # Tăng i lên 1 sau mỗi lần lặp

* Lưu ý while vẫn có thể sử dụng với else



**Vòng lặp for (for loop):**

* **Chức năng:** Lặp đi lặp lại một khối mã lệnh **cho mỗi phần tử trong một chuỗi, danh sách, tuple hoặc các đối tượng iterable khác**.
* **Cấu trúc:**

Python

for biến\_lặp in chuỗi\_hoặc\_danh\_sách:

khối\_lệnh

example:

fruits = ["apple", "banana", "orange"]

for fruit in fruits:

print(fruit)

**Nested loop**

Tại đây sử dụng range(start,end,step)/range(from 0 to [num]) và if để kiểm soát dấu “,”

for i in range(3):

for j in range(2):

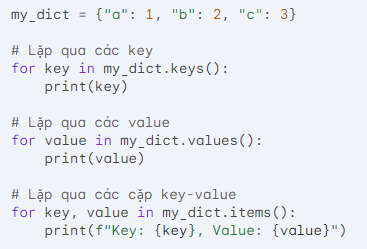
if j < 1:

print(i, j, ",", end=" ")

else:

print(i, j)

**For trong dictionary**

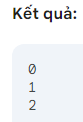


**Break, continue và pass trong Python**

**1. Break:**

* Dùng để thoát khỏi vòng lặp (while, for) hoặc khối lệnh hiện tại.
* Khi gặp break, chương trình sẽ chuyển sang thực thi lệnh tiếp theo sau vòng lặp hoặc khối lệnh.

**Ví dụ:**

Python

for i in range(5):

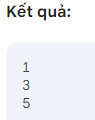
if i == 3:

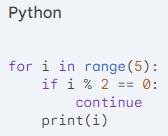
break

print(i)

**2. Continue:**

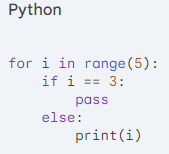
* Dùng để bỏ qua phần còn lại của vòng lặp hiện tại và chuyển sang lần lặp tiếp theo.
* Khi gặp continue, chương trình sẽ không thực thi các lệnh còn lại trong vòng lặp hiện tại.

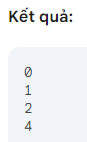
**Ví dụ:**

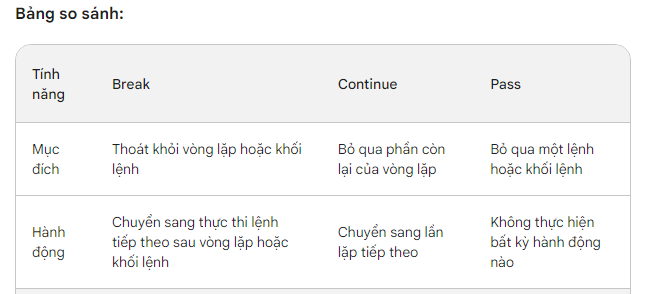


**3. Pass:**

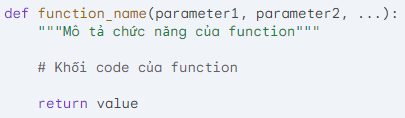
* Dùng để "bỏ qua" một lệnh hoặc khối lệnh.
* Pass không thực hiện bất kỳ hành động nào, nhưng nó có thể được sử dụng để giữ nguyên cấu trúc của chương trình.

**Ví dụ:**





***FUNCTION***



* def: Từ khóa để khai báo function.
* function\_name: Tên của function.
* parameter1, parameter2, ...: Các tham số của function.
* """Mô tả chức năng của function""": Docstring mô tả chức năng của function (không bắt buộc).
* Khối code của function: Các lệnh được thực thi khi gọi function.
* return value: Giá trị trả về của function (không bắt buộc).

!Lưu ý: ~~def sum(arg1 = 1, arg2)~~ phải khai báo cho tất cả parameter phía sau tính từ parameter đầu tiên được khai báo

**Python Modules**

Tạo file mới có tên là [your\_module\_name].py

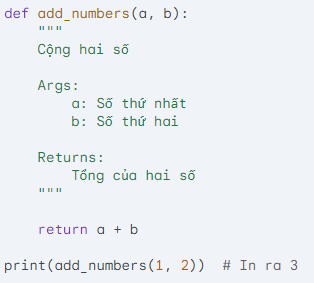
Xây dựng các function

Khi cần sd thì import [your\_module\_name].py

Gọi function từ module: [your\_module\_name].function()

**Docstring là gì?**

**Docstring** là một chuỗi ký tự được sử dụng để mô tả chức năng của một **đối tượng** trong Python, bao gồm

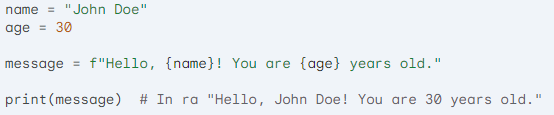


**Có thể dùng dấu + để nối các ký tự**

print("The youngest child is " + child3)

**Trong chuỗi định dạng (f-string):**

* Dùng để **chèn biểu thức** vào chuỗi.
* Ví dụ:

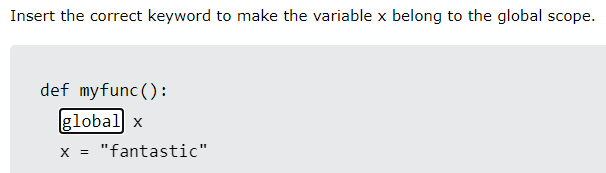


**Trong các hàm built-in:**

* Dùng để **truyền tham số** cho một số hàm built-in.
* Ví dụ:

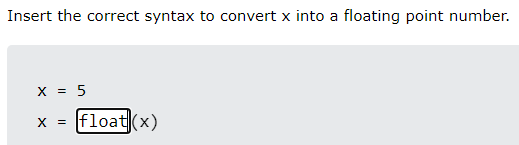


**Global variable**

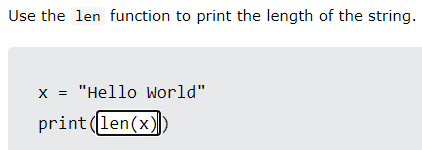


**Data type**: int, str(string), float, list, tuple, dict(dictionary), bool, complex

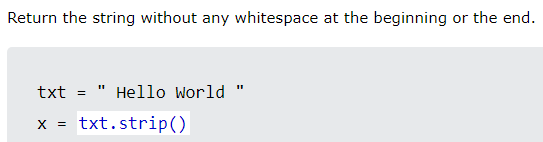
**Convert to another type**



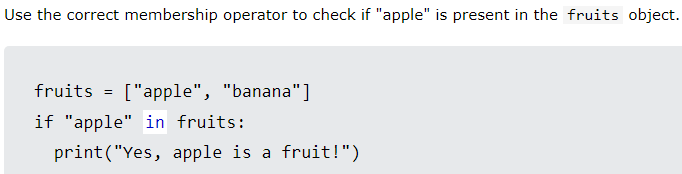
**Length of string**

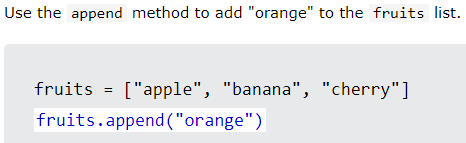


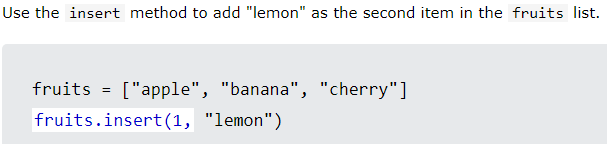
**Remove whitespace**

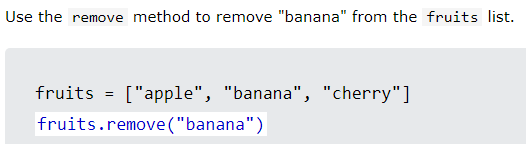


**Check if component available inside dict/tuple/list**

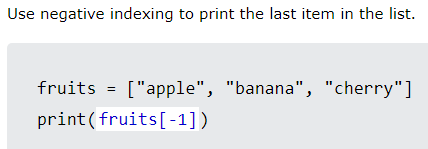


**Add to/remove from list:**

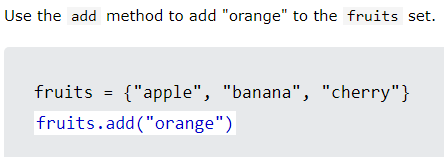
****

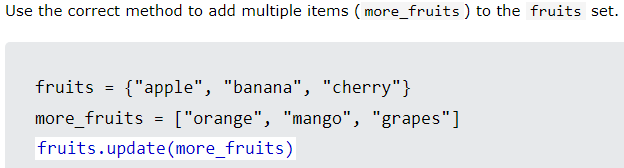
****

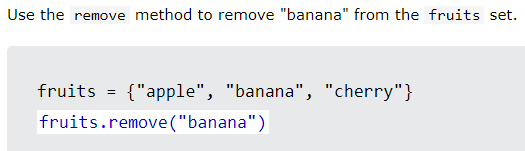
**Print last item using -1 index**

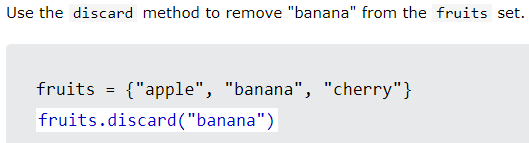
****

**Add to/remove from Sets (single item/list)**

****

****

****

****

**Phân biệt discard và remove trong Python**

**Cả** discard **và** remove **đều là phương thức được sử dụng để loại bỏ phần tử khỏi tập hợp (set) trong Python.** Tuy nhiên, hai phương thức này có một số điểm khác biệt quan trọng:

**1. Xử lý phần tử không tồn tại:**

* discard**:** Nếu phần tử cần loại bỏ không tồn tại trong tập hợp, phương thức discard sẽ **không thực hiện thao tác nào** và **không báo lỗi**.
* remove**:** Nếu phần tử cần loại bỏ không tồn tại trong tập hợp, phương thức remove sẽ **báo lỗi KeyError**.

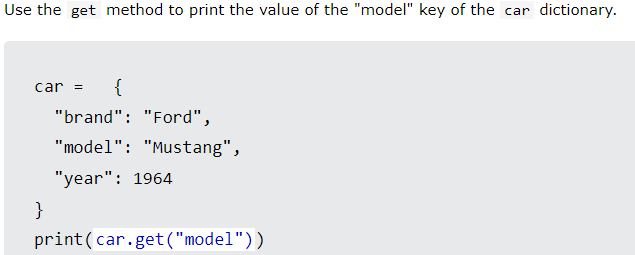
**2. Hiệu suất:**

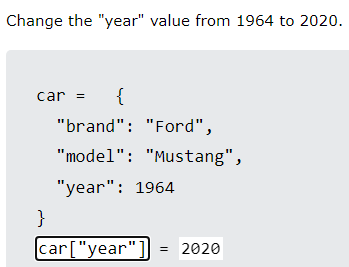
* discard**:** Nhìn chung, discard có **hiệu suất cao hơn** remove vì nó không cần kiểm tra xem phần tử có tồn tại trong tập hợp hay không.
* remove**:** Do phải kiểm tra thêm, remove có thể **chậm hơn** một chút so với discard.

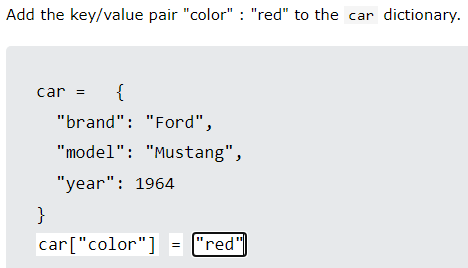
**Kết luận:**

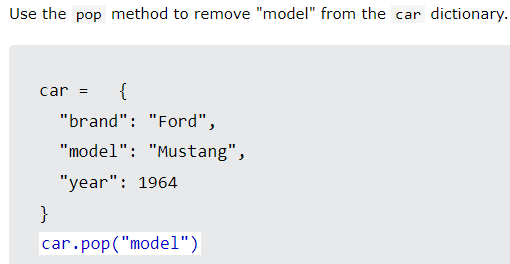
* Nên sử dụng discard khi bạn **không cần quan tâm** đến việc phần tử có tồn tại trong tập hợp hay không.
* Nên sử dụng remove khi bạn **cần đảm bảo** phần tử được loại bỏ khỏi tập hợp và muốn nhận lỗi nếu phần tử không tồn tại.

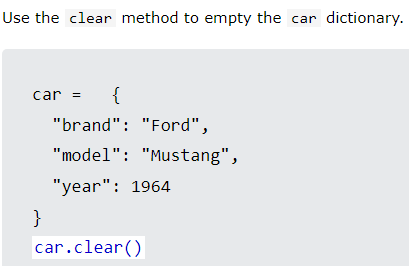
**Dictionary**



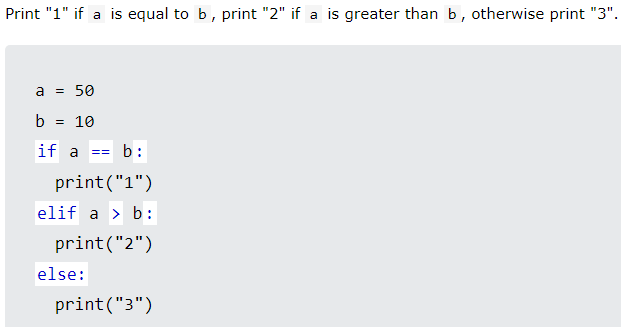


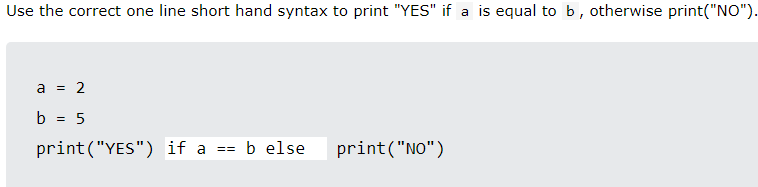




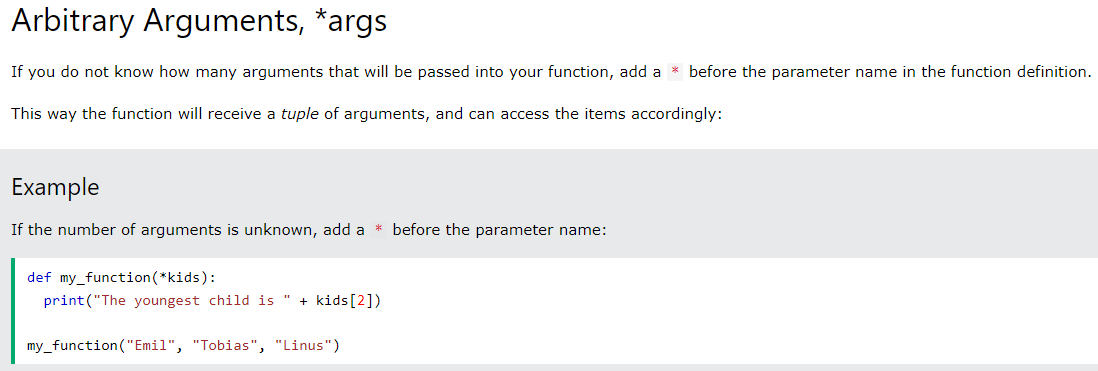


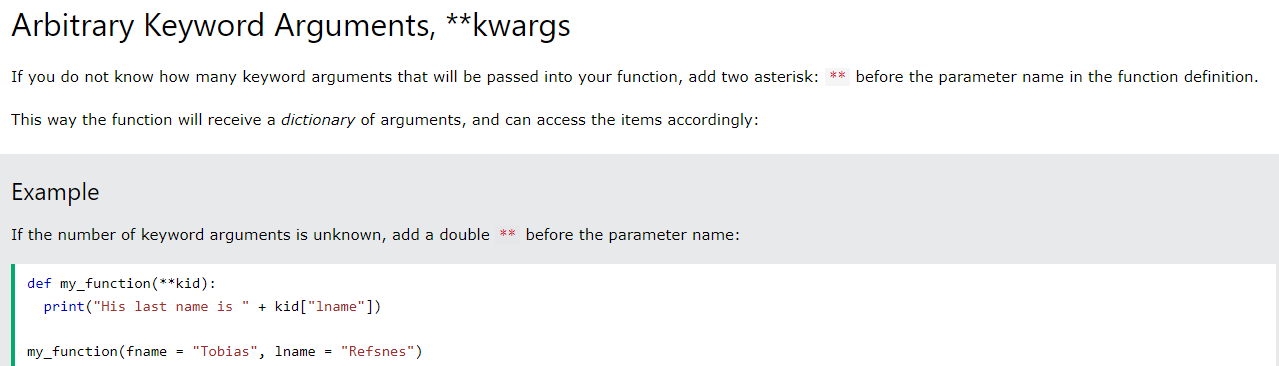
**If…elif…else**





**Function**



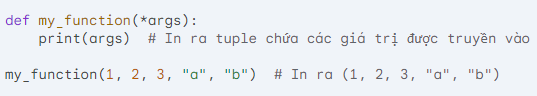


**Ý nghĩa của \* và \*\* trong function**

**1. Dấu sao (\*)**:

**a. Biến số có độ dài thay đổi:**

* Dùng để **thu thập các giá trị còn lại** sau khi đã khai báo các tham số có tên trong định nghĩa hàm.
* Các giá trị được thu thập vào một **tuple**.



**Ý nghĩa của \* và \*\* trong function**

**1. Dấu sao (\*)**:

**a. Biến số có độ dài thay đổi:**

* Dùng để **thu thập các giá trị còn lại** sau khi đã khai báo các tham số có tên trong định nghĩa hàm.
* Các giá trị được thu thập vào một **tuple**.

**Ví dụ:**

Python

def my\_function(\*args):

print(args) # In ra tuple chứa các giá trị được truyền vào

my\_function(1, 2, 3, "a", "b") # In ra (1, 2, 3, "a", "b")

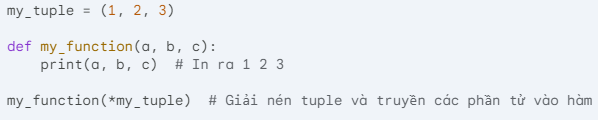
Hãy [thận trọng](https://gemini.google.com/faq#coding) khi sử dụng các đoạn mã.

content\_copy

**b. Giải nén tuple:**

* Dùng để **giải nén các phần tử trong tuple** khi truyền vào hàm.

**Ví dụ:**



**2. Dấu sao kép (**):

**a. Biến số từ khóa có độ dài thay đổi:**

* Dùng để **thu thập các cặp key-value** còn lại sau khi đã khai báo các tham số có tên trong định nghĩa hàm.
* Các giá trị được thu thập vào một **dictionary**.

**Ví dụ:**

def my\_function(\*\*kwargs):

print(kwargs) # In ra dictionary chứa các cặp key-value được truyền vào

my\_function(name="John", age=36, city="Hanoi") # In ra {"name": "John", "age": 36, "city": "Hanoi"}

**b. Giải nén dictionary:**

* Dùng để **giải nén các cặp key-value trong dictionary** khi truyền vào hàm.

my\_dict = {"name": "John", "age": 36}

def my\_function(name, age):

print(name, age) # In ra John 36

my\_function(\*\*my\_dict) # Giải nén dictionary và truyền các cặp key-value vào hàm

**Kết luận:**

* Dấu sao (\*) và dấu sao kép (\*\*) giúp định nghĩa **biến số có độ dài thay đổi** trong hàm.
* Dấu sao (\*) dùng cho **tuple**, dấu sao kép (\*\*) dùng cho **dictionary**.
* Chúng cũng có thể dùng để **giải nén** các cấu trúc dữ liệu khi truyền vào hàm.

**Lambda function**



**Class**

**Cấu trúc của hàm init trong class:**

**Hàm init** là một phương thức đặc biệt được gọi **tự động** khi một **đối tượng mới được tạo** từ lớp tương ứng. Hàm này được sử dụng để **khởi tạo** các thuộc tính (attributes) của đối tượng đó.

**Cấu trúc cơ bản:**

Python

def \_\_init\_\_(self, \*args, \*\*kwargs):

# Khởi tạo các thuộc tính của đối tượng

self.attribute\_1 = value\_1

self.attribute\_2 = value\_2

...

# Xử lý các tham số bổ sung (nếu có)

if args:

# ...

if kwargs:

# ...

**Thành phần:**

1. def \_\_init\_\_(self, \*args, \*\*kwargs)::
   * def: Từ khóa để định nghĩa phương thức.
   * \_\_init\_\_: Tên đặc biệt của phương thức khởi tạo.
   * self: Tham số đầu tiên **bắt buộc** đại diện cho **đối tượng mới được tạo**.
   * \*args: Tham số **tuple** để thu thập các **tham số vị trí** (positional arguments) còn lại.
   * \*\*kwargs: Tham số **dictionary** để thu thập các **tham số có tên** (keyword arguments) còn lại.
2. **Khởi tạo thuộc tính:**
   * Sử dụng self để truy cập và **gán giá trị cho các thuộc tính** của đối tượng.
3. **Xử lý tham số bổ sung (tùy chọn):**
   * Có thể sử dụng args và kwargs để xử lý các tham số bổ sung được truyền vào hàm khởi tạo.

**Ví dụ:**

Python

class Person:

def \_\_init\_\_(self, name, age):

self.name = name

self.age = age

person\_1 = Person("John", 36)

person\_2 = Person("Alice", 28)

print(person\_1.name, person\_1.age) # In ra "John 36"

print(person\_2.name, person\_2.age) # In ra "Alice 28"

Hãy [thận trọng](https://gemini.google.com/faq#coding) khi sử dụng các đoạn mã.

content\_copy

**Lưu ý:**

* Hàm \_\_init\_\_ **không bắt buộc** phải có.
* Nếu không có \_\_init\_\_, các thuộc tính của đối tượng sẽ **không được khởi tạo**.

**Ví dụ về \*args và \*\*kwargs trong class:**

\**1. Ví dụ về args:*

Giả sử ta có một lớp Car với phương thức \_\_init\_\_ nhận thông tin về tên xe và số lượng chỗ ngồi:

Python

class Car:

def \_\_init\_\_(self, name, seats):

self.name = name

self.seats = seats

car\_1 = Car("Toyota Camry", 5)

car\_2 = Car("Honda Civic", 4)

print(car\_1.name, car\_1.seats) # In ra "Toyota Camry 5"

print(car\_2.name, car\_2.seats) # In ra "Honda Civic 4"

Tuy nhiên, nếu ta muốn tạo một chiếc xe với các tùy chọn bổ sung như màu sắc và kiểu động cơ, ta có thể sử dụng \*args để thu thập các tham số vị trí còn lại:

Python

class Car:

def \_\_init\_\_(self, name, seats, \*options):

self.name = name

self.seats = seats

self.options = options

car\_3 = Car("Tesla Model S", 5, "White", "Electric")

print(car\_3.name, car\_3.seats, car\_3.options) # In ra "Tesla Model S 5 ('White', 'Electric')"

\*\*2. Ví dụ về **kwargs:**

Tương tự, ta có thể sử dụng \*\*kwargs để thu thập các tham số có tên còn lại:

Python

class Person:

def \_\_init\_\_(self, name, age, \*\*additional\_info):

self.name = name

self.age = age

self.additional\_info = additional\_info

person\_1 = Person("John", 36, city="Hanoi", occupation="Software Engineer")

print(person\_1.name, person\_1.age, person\_1.additional\_info) # In ra "John 36 {'city': 'Hanoi', 'occupation': 'Software Engineer'}"

\*\*Kết hợp \*args và **kwargs:**

Ta có thể kết hợp cả \*args và \*\*kwargs trong cùng một phương thức:

Python

class MyClass:

def \_\_init\_\_(self, \*args, \*\*kwargs):

self.args = args

self.kwargs = kwargs

my\_object = MyClass(1, 2, 3, name="John", age=36)

print(my\_object.args, my\_object.kwargs) # In ra ((1, 2, 3), {'name': 'John', 'age': 36})

**Lưu ý:**

* Thứ tự của \*args và \*\*kwargs trong định nghĩa phương thức là quan trọng.
* Khi sử dụng \*args hoặc \*\*kwargs, cần đảm bảo xử lý các giá trị được thu thập một cách phù hợp.

**Kết luận:**

\*args và \*\*kwargs là hai công cụ hữu ích giúp thu thập các tham số được truyền vào phương thức trong class Python, giúp cho việc viết code linh hoạt và dễ dàng hơn.

**Inheritance**

Để giữ lại \_\_init\_\_ của parent class, sử dụng cú pháp:  
class Student(Person):

def \_\_init\_\_(self, fname, lname):

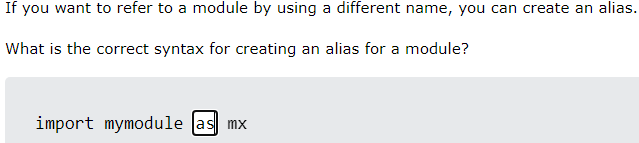
Person.\_\_init\_\_(self, fname, lname) hoặc super().\_\_init\_\_(fname, lname)

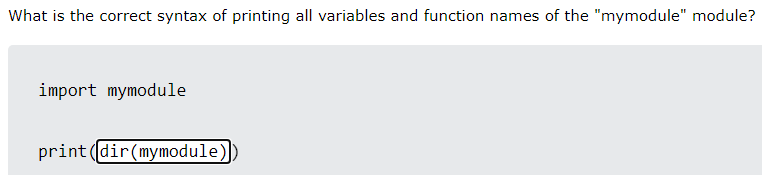
self.graduationyear = year #add properties, same collumn with above line

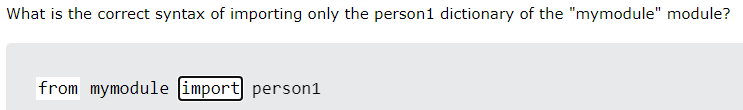
def welcome(self): #add methods

print("Welcome", self.firstname, self.lastname, "to the class of", self.graduationyear)

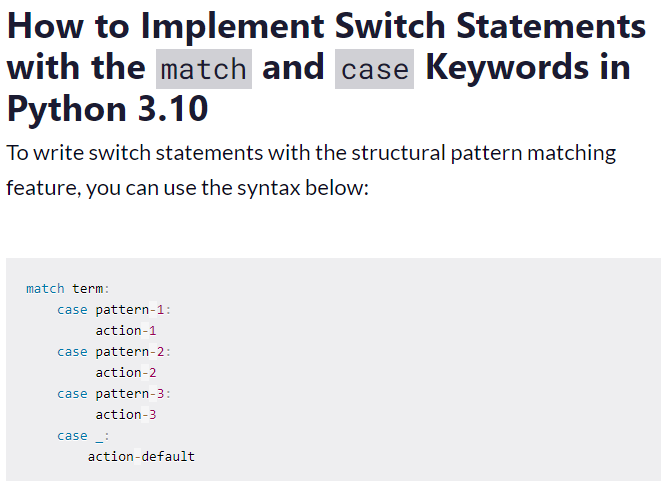
**Module**







**Match – case (switch case replacement)**



**Os, platform, stat and time library**

**1.** platform.system()**:**

Lệnh này trả về tên hệ điều hành đang chạy chương trình. Ví dụ: trên Windows, nó sẽ trả về "Windows".

**2.** platform.uname()**:**

Lệnh này trả về một tuple chứa thông tin về hệ thống, bao gồm:

* Tên hệ điều hành
* Tên máy chủ
* Phiên bản hệ điều hành
* Kiến trúc CPU
* Cờ hiệu hệ điều hành

**3.** os.getcwd()**:**

Lệnh này trả về thư mục làm việc hiện tại của chương trình.

**4.** os.listdir(path)**:**

Lệnh này trả về danh sách các file và thư mục trong thư mục được chỉ định bởi path.

**5.** os.stat(path)**:**

Lệnh này trả về một object stat chứa thông tin về file hoặc thư mục được chỉ định bởi path.

**6.** info.st\_size**:**

* Hàm này trả về kích thước của tệp được tính bằng byte.
* Nó cho biết tệp chiếm bao nhiêu không gian lưu trữ trên thiết bị.

**7.** stat.filemode(info.st\_mode)**:**

* Hàm này trả về quyền truy cập của tệp dưới dạng chuỗi có thể đọc được.
* Ví dụ, nếu kết quả là '-rw-r--r--', nghĩa là:
  + -: Là dấu gạch nối, không có nghĩa cụ thể.
  + r: Chủ sở hữu tệp có quyền đọc.
  + w: Chủ sở hữu tệp có quyền ghi.
  + -: Nhóm sở hữu tệp không có quyền truy cập.
  + r: Người dùng khác (không phải chủ sở hữu hoặc nhóm) có quyền đọc.
  + -: Người dùng khác không có quyền ghi.
  + r: Người dùng khác có quyền thực thi (nếu là tệp thực thi).

**8.** info.st\_uid**:**

* Hàm này trả về ID người dùng (user ID) của chủ sở hữu tệp.
* Nó là một số nguyên duy nhất xác định người dùng trên hệ thống.

**9.** info.st\_dev**:**

* Hàm này trả về ID thiết bị (device ID) nơi tệp được lưu trữ.
* Nó là một số nguyên duy nhất xác định thiết bị lưu trữ trên hệ thống.

**10.** time.ctime(info.st\_ctime)**:**

* Hàm này trả về thời gian tạo tệp dưới dạng chuỗi định dạng dễ đọc (ví dụ, "Thứ Năm, 23 Tháng Hai 2024 09:35:42 +0700").
* info.st\_ctime là thời gian tính bằng giây kể từ 00:00:00 UTC ngày 1 tháng 1 năm 1970.

**11.** time.ctime(info.st\_mtime)**:**

* Hàm này trả về thời gian tệp được sửa đổi cuối cùng dưới dạng chuỗi định dạng dễ đọc.
* info.st\_mtime là thời gian tính bằng giây kể từ 00:00:00 UTC ngày 1 tháng 1 năm 1970.

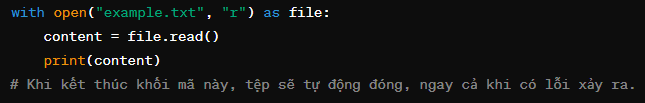
**12.** time.ctime(info.st\_atime)**:**

* Hàm này trả về thời gian tệp được truy cập cuối cùng dưới dạng chuỗi định dạng dễ đọc.
* info.st\_atime là thời gian tính bằng giây kể từ 00:00:00 UTC ngày 1 tháng 1 năm 1970.

**File**

Sử dụng with (để khỏi xài close) trước open(“file\_name”,’permissions like r, r+, w, w+’) as sylas:

Do something



**Tách phần tử ra để tạo list mới**

Trong Python, word[1] for word in words là một cấu trúc của list comprehension. List comprehension là một cách ngắn gọn để tạo ra danh sách trong Python.

Trong trường hợp này, words là một danh sách các bộ giá trị, và word là một biến tạm thời đại diện cho mỗi bộ giá trị trong danh sách khi lặp qua nó. word[1] lấy phần tử thứ hai trong mỗi bộ giá trị (vì chỉ số trong Python bắt đầu từ 0).

Ví dụ, nếu words là [(1, 'a'), (2, 'b'), (3, 'c')], thì word[1] for word in words sẽ tạo ra danh sách ['a', 'b', 'c'].

**Regular expression**

<https://www.regular-expressions.info/quickstart.html>

<https://toidicode.com/regular-expression-trong-python-365.html>

<https://toidicode.com/regular-expression-trong-python-p2-366.html>

<https://regex101.com/>

re.match()

Hàm re.search trong module re của Python trả về một đối tượng match nếu tìm thấy mẫu regex trong chuỗi được cung cấp, hoặc None nếu không tìm thấy.

**Đối tượng** match **chứa các thông tin sau:**

* **Vị trí bắt đầu và kết thúc của chuỗi khớp:** Thuộc tính .start() và .end() cung cấp vị trí bắt đầu và kết thúc của chuỗi khớp trong chuỗi ban đầu.
* **Nhóm con:** Các nhóm con trong mẫu regex có thể được truy cập thông qua các thuộc tính .group(), .groups() và .groupdict().
* **Toàn bộ chuỗi khớp:** Thuộc tính .string cung cấp toàn bộ chuỗi ban đầu được sử dụng cho quá trình tìm kiếm.

**Ví dụ:**

Python

import re

text = "Đây là một ví dụ về sử dụng re.search"

# Tìm kiếm mẫu "re.search"

match = re.search(r"re\.search", text)

if match:

# In vị trí bắt đầu và kết thúc của chuỗi khớp

print(f"Vị trí bắt đầu: {match.start()}")

print(f"Vị trí kết thúc: {match.end()}")

# In toàn bộ chuỗi khớp

print(f"Chuỗi khớp: {match.string}")

else:

print("Mẫu regex không được tìm thấy")

**Kết quả:**

Vị trí bắt đầu: 13

Vị trí kết thúc: 23

Chuỗi khớp: Đây là một ví dụ về sử dụng re.search

**Ngoài ra, bạn có thể sử dụng các cờ sau với hàm** re.search**:**

* re.IGNORECASE**:** Bỏ qua việc phân biệt chữ hoa và chữ thường khi tìm kiếm.
* **re.MULTILINE`:** Cho phép ^ và $ khớp với đầu và cuối của mỗi dòng trong chuỗi.
* **re.DOTALL`:** Cho phép . khớp với tất cả các ký tự, bao gồm cả dòng mới.

**Lưu ý:**

* Hàm re.search chỉ tìm kiếm lần xuất hiện đầu tiên của mẫu regex trong chuỗi.
* Để tìm kiếm tất cả các lần xuất hiện của mẫu regex, hãy sử dụng hàm re.findall.

Hy vọng giải thích này giúp bạn hiểu rõ hơn về cách thức hoạt động của hàm re.search!

**sub(pattern, replace, string, flags)**

Phương thức này có tác dụng so khớp và thay thế chuỗi so khớp được.

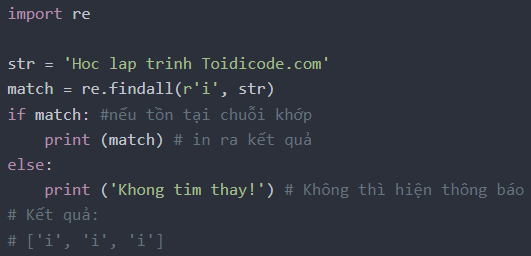
**Trong đó**:

* pattern là chuỗi regular expression.
* replace là chuỗi thay thế cho các chuỗi được so khớp.
* string là chuỗi cần so khớp.
* flags là các kiểu so khớp mà bạn muốn thực hiện (xem ở dưới). Nếu để trống thì flags sẽ bằng 0.



**findall(partern, string, flags)**

Phương thức này có tác dụng so khớp và trả về tất cả các chuỗi mà nó đã so khơp được, còn lại các tham số truyền và sử dụng như đối với phương thức search.



* **Biểu thức đặc biệt:**
  + \*: Lặp lại ký tự trước đó 0 lần hoặc nhiều lần
  + +: Lặp lại ký tự trước đó 1 lần hoặc nhiều lần
  + ?: Lặp lại ký tự trước đó 0 lần hoặc 1 lần
  + []: Khớp với bất kỳ ký tự nào trong tập hợp
  + [^]: Khớp với bất kỳ ký tự nào không nằm trong tập hợp
  + |: Dấu hoặc (hoặc)

**I for i in list**

Được xài để duyệt và trả về tất cả phần tử trong danh sách (có thể kèm điều kiện)

filtered = [i for i in my\_list if i % 2]

Here the important part is the iteration and the filtering, ie iterating through a list of integers and selecting just the even ones. The i for i part is just saying don't do anything else to the item, just include it in the output list directly.

Of course you *can* do something to the item, such as:

filtered = [i\*2 for i in my\_list if i % 2]

where you're taking those even numbers and doubling them.

**Xóa trùng lập trong list**

Bằng cách sử dụng set ta có thể xóa các trùng lặp trong list

**Unpack generator \***

**A screenshot of a computer

Description automatically generated**

**A screenshot of a computer

Description automatically generated**

**A screenshot of a computer program

Description automatically generated**

**A screen shot of a computer code

Description automatically generated**

**Find trong string**

Nếu ta có 2 array a và b, muốn tìm array a trong b thì sử dụng b.find(a)