Getting Started – Bắt Đầu:

1. Python Có Phân Biệt Chữ Hoa Và Chữ Thường Không?

* Có

1. Cách Xem Phiên Bản Của 1 Thư Viện?

* Vào CMD, gõ

pip3 show <Tên Thư Viện>

* Ví dụ

pip3 show tensorflow

1. Hiện Mô Tả Cách Sử Dụng 1 Object?

help(<Object>)

* Ví dụ

help(print)

1. Show Phiên Bản Python?

* Vào CMD, gõ

python -V

1. Cập Nhật Pip?

* Chạy lệnh sau trong CMD

pip3 -m pip3 install -U pip3

1. Pip Với Pip3?

* Pip tải thư viện cho Python 2, còn Pip3 tải thư viện cho Python 3

1. Cách Cài Đặt 1 Thư Viện?

pip3 install --default-timeout=<Số Giây> <Tên Thư Viện>

* Mặc định

|  |  |
| --- | --- |
| <Số Giây> | 15 |

* Nếu thời gian cài đặt vượt quá <Số Giây> thì báo lỗi
* Ví dụ

pip3 install --default-timeout=100 matplotlib

Data Type – Kiểu Dữ Liệu:

1. Giá Trị Vô Cực?

float("inf")

1. Biểu Diễn Giá Trị Nguyên Bằng Nhị Phân?

* Thêm 0b đằng trước chuỗi nhị phân
* Ví dụ, lệnh sau sẽ trả về 5

0b101

1. Chuyển String Thành Số Nguyên?

* Lệnh sau sẽ trả về số nguyên từ 1 String, từ hệ cơ số <Hệ Cơ Số> thành hệ cơ số 10

int(<String>, <Hệ Cơ Số>)

* Ví dụ

foo = int("45123", 6)

* Ta có

|  |  |
| --- | --- |
| foo | 6315 |

File – Tệp:

1. Cách Hoạt Động?

* Khi chương trình kết thúc, mọi File Object đang mở đều đóng
* Khi File Object đóng rồi, thì bạn không thể thực hiện thao tác ghi, đọc File ứng với File Object này thông qua File Object này cho dù File đó có đang mở, nhưng vẫn có thể thông qua File Object khác
* Khi mở 1 File trong chế độ đọc, và sau đó sử lệnh đọc ít nhất 1 kí tự, thì dữ liệu từ File đó thời điểm hiện tại mới được Copy vào bộ đệm mới, và trong suốt quá trình sau đó cho đến khi File Object đóng, dữ liệu trong bộ đệm này không đổi
* Khi mở 1 File trong chế độ ghi, thì ngay lập tức File đó sẽ bị làm rỗng, bộ đệm ban đầu cũng rỗng, khi bạn thực hiện thao tác ghi, thực chất là làm trên bộ đệm, khi File Object ứng với File này đóng, thì nội dung đã ghi trong bộ đệm sẽ đè lên File thật, ví dụ trong File thật hiện tại là abcde, trong bộ đệm là 123, thì khi bộ đệm đè lên File thật sẽ thành 123de
* 1 File đang mở thì có thể có nhiều File Object ứng với nó, mỗi cái sẽ có 1 bộ đệm và con trỏ độc lập
* 1 File đã đóng có thể mở lại
* Vị trí con trỏ của File Object khi mới mở là kí tự đầu tiên của File
* Vị trí cuối cùng mà con trỏ có thể đi tới là kế ngay sau kí tự cuối cùng của File, gọi vị trí này là Max Position

1. Mở 1 File Và Gán Nó Vào 1 File Object?

<File Object> = open(<Đường Dẫn Tới File Có Phần Mở Rộng>, <Chế Độ>)

* Mặc định

|  |  |
| --- | --- |
| <Chế Độ> | "r" |

* Các <Chế Độ> khác nhau

|  |  |
| --- | --- |
| <Chế Độ> | Các thao tác được làm |
| "r" | Đọc |
| "w" | Ghi |
| "a" | Ghi, khi mở File không bị làm rỗng, đồng thời con trỏ luôn nằm tại Max Position hiện tại, không thể thay đổi vị trí con trỏ theo ý muốn |
|  |  |

* Ví dụ

foo = open("bar.txt", "w")

* Để đóng 1 File Object

<File Object>.close()

* Để không phải viết lệnh mở đóng cho mệt

with open(<Đường Dẫn Tới File Có Phần Mở Rộng>, <Chế Độ>) as <File Object>:

<Làm Gì Đó>

* Tương đương

<File Object> = open(<Đường Dẫn Tới File Có Phần Mở Rộng>, <Chế Độ>)

<Làm Gì Đó>

<File Object>.close()

* Để trả về <Đường Dẫn Tới File Có Phần Mở Rộng>

<Đường Dẫn Tới File Có Phần Mở Rộng> = <File Object>.name

* Để trả về <Chế Độ>

<Chế Độ> = <File Object>.mode

* Để Check xem 1 File Object đóng chưa

<Đóng Chưa> = <File Object>.closed

* <Đóng Chưa> = True nếu đóng rồi, còn không thì = False

1. Đọc File?

* Trả về chuỗi kí tự của File, xuống dòng cũng tính là 1 kí tự, bắt đầu từ vị trí hiện tại của con trỏ, lấy khoảng <Số Kí Tự> kí tự, đồng thời đưa con trỏ tới vị trí kế ngay sau kí tự cuối cùng của chuỗi kí tự

<Chuỗi Kí Tự> = <File Object>.read(<Số Kí Tự>)

* Mặc định

|  |  |
| --- | --- |
| <Số Kí Tự> | Vừa đủ để con trỏ tới Max Position |

* Nếu <Số Kí Tự> làm cho con trỏ vượt quá vị trí cuối cùng thì sẽ bị Clip để con trỏ vừa đủ tới vị trí cuối cùng
* Ví dụ trong File ứng với File Object foo có nội dung sau, vị trí hiện tại của con trỏ là chỗ in đậm

foo b**a**r bar

bob alice 123.

* Ta viết

far = foo.read(8)

bar = foo.read()

* Ta có

|  |  |
| --- | --- |
| far | "ar bar\nb" |
| bar | "ob alice 123." |

* Cũng tương tự như trên, nhưng <Số Kí Tự> sẽ bị Clip sao cho vị trí con trỏ sau khi đọc không vượt quá vị trí kế ngay sau kí tự xuống dòng đầu tiên gặp phải đi từ vị trí hiện tại, gọi vị trí này là Max Line Position

<Chuỗi Kí Tự> = <File Object>.readline(<Số Kí Tự>)

* Mặc định

|  |  |
| --- | --- |
| <Số Kí Tự> | Vừa đủ để con trỏ tới Max Line Position |

* Để trả về 1 List có các phần tử lần lượt là các dòng trong File

<Các Dòng> = <File Object>.readlines(<Số Kí Tự>)

* Mặc định

|  |  |
| --- | --- |
| <Số Kí Tự> | Vừa đủ để con trỏ tới Max Position |

* Cơ chế như sau
* Nếu vị trí theo lý thuyết của con trỏ sau khi đọc ứng với <Số Kí Tự> nằm ngay tại kí tự xuống dòng thì vị trí thật của nó sẽ kế ngay bên phải kí tự xuống dòng này
* Nếu không thì vị trí thật sẽ = vị trí theo lý thuyết tăng dần cho đến khi nằm kế bên phải kí tự xuống dòng tiếp theo
* Trích xuất chuỗi kí tự bắt đầu từ vị trí hiện tại tới kí tự ngay trước vị trí thật
* Cắt chuỗi này tại các điểm ngay bên phải các kí tự xuống dòng rồi đưa nó vào List
* Ví dụ trong File ứng với File Object foo có nội dung sau, vị trí hiện tại của con trỏ là chỗ in đậm

day **l**a dong 1.

day la dong 2.

day la dong 3.

day la dong 4.

* Ta viết

far = foo.readlines(30)

* Ta có

|  |  |
| --- | --- |
| far | ['la dong 1.\n', 'day la dong 2.\n', 'day la dong 3.\n'] |

* Ta cũng có thể lặp qua các dòng

for <Dòng> in <File Object>:

<Làm Gì Đó>

* Tương đương lặp khối Code sau cho đến khi con trỏ tới Max Position

<Dòng> = <File Object>.readline()

<Làm Gì Đó>

1. Trả Về Vị Trí Hiện Tại Của Con Trỏ?

* Quy ước vị trí của kí tự đầu tiên trong File là 0, cứ sau 1 kí tự thì vị trí + 1, sau kí tự \n tức kí tự xuống dòng sẽ tính là + 2

<Vị Trí> = <File Object>.tell()

1. Di Chuyển Con Trỏ Đến Vị Trí Chỉ Định?

* Quy ước như cũ, tuy nhiên khi ta chủ động thiết lập vị trí cho con trỏ, thì nó có thể > Max Position, hành vi ở vị trị ngoài lề này = hành vi ở Max Position
* Nếu di chuyển vào vị trí kí tự \ hay n của \n thì khi đọc từ vị trí này, vẫn đọc nguyên \n và coi nó là 1 kí tự

<File Object>.seek(<Vị Trí Mới>)

* Ví dụ

foo.seek(10)

1. Ghi Vào File?

* Theo như cơ chế thì ta đang ghi vào bộ đệm
* Để ghi đè 1 chuỗi kí tự vào bộ đệm, bắt đầu từ vị trí con trỏ, đồng thời đưa con trỏ tới kế ngay sau kí tự cuối cùng của <Chuỗi Kí Tự>, \n được tính là 2 kí tự khi ghi đè, do đó cần 2 kí tự để lấp kí tự này, và kí tự này có thể lấp 2 kí tự, nếu chỉ lấp được \ hoặc n của kí tự này thì phần còn lại sẽ được tái sinh thành \n hoàn chỉnh

<File Object>.write(<Chuỗi Kí Tự>)

* Ví dụ trong File ứng với File Object foo có nội dung sau, vị trí hiện tại của con trỏ là chỗ in đậm

foo b**a**r bar

bob alice 123.

* Ta viết

foo.write("toidangbucu")

* Ta có nội dung mới

foo btoidangbucu alice 123.

Package – Gói:

1. Cấu Trúc 1 Package?

* Package thực chất là 1 thư mục chứa File “\_\_init\_\_.py”, cùng với các File và thư mục khác, mỗi thư mục trong Package cũng phải là 1 Package

1. Cách Import 1 Module?

* Khi Import 1 File nào đó, giả sử B vào File A, thì B sẽ được chạy trong A ngay lập tức với tư cách là 1 Module
* Khi 1 File được chạy với tư cách là 1 Module, thì sẽ có 1 thư mục tên “\_\_pycache\_\_” được tạo ngay bên cạnh File đó nếu chưa tồn tại, nó sẽ chứa Byte Code của File này
* Đồng thời, khi ta chạy 1 File với tư cách là 1 Module, ban đầu Python sẽ dò xem ở thư mục hiện tại, đã có File này hay chưa, nếu chưa thì tiếp tục dò trong Python Path, nếu không có nữa thì báo lỗi
* Để chạy Module nhanh hơn, Python sẽ ưu tiên chạy Byte Code tương ứng trong thư mục “\_\_pycache\_\_” trước, nếu nó tồn tại và hợp lệ, Byte Code là hợp lệ khi thời điểm chỉnh sửa cuối cùng của nó và thời điểm chỉnh sửa cuối cùng của File gốc = nhau, hoặc mã Hash khớp với nhau
* Khi chạy 1 File không với tư cách là 1 Module, thì thuộc tính “\_\_name\_\_” của File này = “\_\_main\_\_”, còn nếu là 1 Module, thì thuộc tính “\_\_name\_\_” của nó là tên Module, bạn Import Module kiểu nào thì tên của nó y chang như cách bạn Import
* Ví dụ
* Trong thư mục “father”, chứa Package “foo” và File “child.py”, trong “foo” chứa Package “bar” và File “fool.py”, trong “bar” chứ File “boo.py”
* Tại File “child.py”

from foo.bar import boo

* Tại File “boo.py”

print(\_\_name\_\_)

* Khi chạy File “child.py”, Terminal sẽ in ra “foo.bar.boo”, đây chính là tên Module
* Khi Import 1 Module của 1 Package, thì những File “\_\_init\_\_.py” trên đường dẫn tới Module đó trong Package cũng sẽ được chạy với tư cách là Module
* Ta chỉ có thể sử dụng Relative Import trong 1 File khi nó đã được chạy với tư cách là 1 Module, Relative Import sử dụng dấu “.” để lên 1 bậc hoặc xuống 1 bậc
* Tiếp tục với ví dụ trên
* Tại File “boo.py”

from .. import fool

* Câu lệnh trên chỉ chạy thành công nếu bạn chạy File “child.py”, vì mỗi dấu “.” trong “..” sẽ đưa tên Module lên 1 bậc, nghĩa là câu lệnh trên import module tên “foo.fool”, điều này cho thấy số dấu “.” khi Import phải = số dấu “.” trong tên Module
* Nếu đã Import C từ B vào A, thì ta cũng có thể Import C từ A, thay vì C từ B

1. Tạo File Wheel?

* File Wheel thực chất dùng để cài đặt các Package có trong nó, và khi cài đặt nó sẽ được đưa vào Python Path, nó bản chất là 1 thư viện
* Để tạo File Wheel, trong thư mục chứa các Package của bạn, Copy đoạn Code sau vào 1 File

from setuptools import setup, find\_packages

setup(

name = <Tên Thư Viện>,

version = <Version>,

packages = find\_packages()

)

* <Tên Thư Viện>, <Version> muốn đặt kiểu gì cũng được
* Chạy File trên với tham số “bdist\_wheel” bằng CMD
* Ví dụ

python setup.py bdist\_wheel

* Sau khi chạy, sẽ có 1 số thư mục được tạo bên cạnh File bạn vừa chạy, trong đó có thư mục “dist”, trong đó có File Wheel
* Để cài đặt 1 File Wheel, nói cách khác là cài đặt thư viện, nhập vào CMD

pip install <Đường Dẫn Tới File Wheel>

* Ví dụ

pip install dist\conchothien-9.9-py3-none-any.whl

* Sau khi cài đặt, bạn có thể sử dụng thư viện này ở mọi nơi

String – Xâu:

1. String Nội Suy?

<String Nội Suy> = f<String Chứa Biến>

* Muốn nội suy thì đặt biến trong ngoặc nhọn
* Ví dụ

foo = f"abc{bar}def"

* Ta có

|  |  |
| --- | --- |
| bar | 123 |
| foo | abc123def |

* Để in dấu ngoặc nhọn khi nội suy dùng 2 lần ngoặc nhọn
* Ví dụ

foo = f"ab}}c{{def"

* Ta có

|  |  |
| --- | --- |
| foo | ab}c{def |

1. Cách Viết Xuống Dòng Xâu Dài Vãi Lồn Trong Python Nhưng Khi In Thì Không Xuống Dòng?

<Biến> = "<Dòng 1>\

<Dòng 2>\

<Dòng 3>\

…

<Dòng N>"

* Ví dụ

text = “foo is bar and bar is not foo \

but why foo is foo \

and not bar is foo”

print(text)

* Màn hình CMD

foo is bar and bar is not foo but why foo is foo and not bar is foo

1. Biểu Diễn String Dưới Dạng Thô Hiện Các Kí Tự Thoát?

<String Thô> = repr(<String>)

* Ví dụ

foo = repr(bar)

* Ta có

|  |  |
| --- | --- |
| bar | "con cu cac\nthat la to" |
| foo | "'con cu cac\\nthat la to'" |

* Khi in ra bar

con cu cac

that la to

* Khi in ra foo

'con cu cac\nthat la to'

1. Cách Tự Động Thêm Các Kí Tự Thoát Để Định Dạng Đúng Như Những Gì Mình Nhìn Thấy Tại Code Editor?

<String Đã Thêm Kí Tự Thoát> = '''<Dòng 1>

<Dòng 2>

<Dòng 3>

…

<Dòng N>'''

* Ví dụ

foo = '''toi la thang cho

cua cong san

dit me toi'''

* Khi in foo

toi la thang cho

cua cong san

dit me toi

1. Cách Biến Xâu Thành Tập Hợp Các Kí Tự Xuất Hiện Trong Xâu Đó?

<Tập Hợp Các Kí Tự> = set(<Xâu>)

* Dấu cách cũng tính
* Ví dụ

foo = set(“b/ar 12a nứ”)

* foo là {‘ ‘, ‘/’, ‘1’, ‘2’, ‘b’, ‘a’, ‘r’, ‘n’, ‘ứ’}

1. Cách Sắp Xếp 1 Mảng Chỉ Toàn Xâu Theo Từ Điển?

<Mảng Đã Sắp Xếp> = sorted(<Mảng Chưa Sắp Xếp>)

* Lưu ý thứ tự từ điển là dấu cách xếp đầu, đến kí tự đặc biệt, số, chữ in hoa, chữ in thường, chữ có dấu
* Ví dụ

foo = sorted([‘a’, ‘ab’, ‘ca’, ‘1’, ‘ ‘, ‘/’, ‘ứ’])

* foo là [‘ ’, ‘/’, ‘1’, ‘a’, ‘ab’, ‘ca’, ‘ứ’]

1. Cách Chuyển Đối Tượng Iterable Chứa Các Phần Tử Toàn Là Xâu Thành 1 Xâu Duy Nhất?

<Xâu Nối> = <Xâu Ngăn Cách>.join(<Đối Tượng Iterable Chứa Toàn Xâu>)

* Ví dụ
* x là [‘foo’, ‘bar’, ‘i’, ‘ stupid’]

y = ‘shit’.join(x)

* “y” là “fooshitbarshitishitstupid”

1. Cách Viết Xâu Không Có Nội Suy Kí Tự Đặc Biệt?

r<Xâu Chứa Kí Tự Đặc Biệt>

* Ví dụ

foo = r"huhu.\n"

* “foo” sẽ có giá trị là “huhu.\n”, kí tự đặc biệt “\n” được xem là kí tự thường

1. Byte String?

* Tạo trực tiếp 1 Byte String

<Byte String> = b<String>

* Ví dụ

foo = b"bar"

* Tạo gián tiếp 1 Byte String

<Byte String> = <String>.encode(<Kiểu Mã Hóa>)

* Ví dụ

foo = "bar".encode("ascii")

* Cách đưa 1 Byte String trở lại thành String

<String> = <Byte String>.decode(<Kiểu Mã Hóa>)

* Encode = kiểu mã hóa nào thì Decode = kiểu mã hóa đó
* Ví dụ

boo = b"bar".decode("utf-8")

* Chỉ có Byte String mới được lưu vào File, String không được
* Mỗi File hình ảnh bản chất là 1 Byte String

1. Thay 1 Chuỗi Kí Tự Trong 1 String Thành Chuỗi Kí Tự Khác?

<String Mới> = <String>.replace(<Chuỗi Kí Tự Gốc>, <Chuỗi Kí Tự Mới>)

* Dò từ trái sang phải, và thế tất cả
* Ví dụ

foo = bar.replace(".h", "abc")

* Ta có

|  |  |
| --- | --- |
| bar | "concucac.h.h” |
| foo | "concucacabcabc" |

1. Cách Tạo User Input?

<User Input> = input(<Yêu Cầu>)

* <Yêu Cầu> là 1 dòng chữ sẽ hiện lên, như để nhắc người dùng nhập <Input> chẳng hạn
* Chương trình sẽ tạm dừng cho đến khi người dùng nhập <Input> rồi nhấn “Enter”, giá trị của <User Input> = <Input> mà người dùng nhập
* Ví dụ

foo = input("Mày tên là gì: ")

* Ta có

|  |  |
| --- | --- |
| foo | "Tokuda" |

1. In Ra Màn Hình Nhưng Không Xuống Dòng?

print(<String>, end = <Chuỗi Cuối>)

* Mặc định

|  |  |
| --- | --- |
| <Chuỗi Cuối> | "\n" |

* Ví dụ

print("vaicalon", end = "ditme")

print("vaicalcac", end = "ditcha")

* Màn hình CMD

vaicalonditmevaicacacditcha

1. Check Xem 1 String Có Chứa 1 Đoạn String Con?

* Lệnh sau sẽ trả về True nếu <String> chứa <String Con>

<String Con> in <String>

* Ví dụ

foo = "abc" in "deabc123"

* Ta có

|  |  |
| --- | --- |
| foo | True |

1. Check Xem 1 String Có Bắt Đầu Bằng 1 Đoạn String Con?

* Lệnh sau sẽ trả về True nếu <String> bắt đầu bằng <String Con>

<String>.startswith(<String Con>)

* Ví dụ

foo = "abcdef".startswith("abc")

* Ta có

|  |  |
| --- | --- |
| foo | True |

List – Dãy:

1. Cách Chèn 1 Phần Tử Vào Cuối List?

<List>.append(<Phần Tử>)

1. Cách Chèn Nhiều Phần Tử Vào Cuối List?

<List>.append(<List Các Phần Tử>)

* Ví dụ

bar.append([1, [2, 3], "abc"])

* Ta có

|  |  |
| --- | --- |
| bar trước | [9, 8, 7] |
| bar sau | [9, 8, 7, 1, [2, 3], "abc"] |

1. Cách Lặp 1 Lúc Qua 2 Đối Tượng Iterable?

for <Phần Tử Của A>, <Phần Tử Của B> in zip(<A>, <B>):

<Làm Gì Đó Với Phần Tử Của A Và B>

* “zip(<A>, <B>)” giống như ma trận với dòng 1 là A và dòng 2 là B sau đó chuyển vị

1. Quy Tắc Tạo Đối Tượng Iterable Con?

* Cho A là 1 đối tượng Iterable, để trả về các phần tử có số thứ tự từ a đến b, viết

A[a : b + 1]

1. Trả Về Phiên Bản Đảo Chiều 1 Iterable?

<Iterable>[::-1]

* Ví dụ

foo = bar[::-1]

* Ta có

|  |  |
| --- | --- |
| bar | [1, 2, 3, 4] |
| foo | [4, 3, 2, 1] |

1. Cách Nối Nhiều List?

<List Tổng Hợp> = <List 1> + <List 2> + …

* Ví dụ

foo = [1, 2, 3] + [4, 5] + [[4, 8], "a"]

* Ta có

|  |  |
| --- | --- |
| foo | [1, 2, 3, 4, 5, [4, 8], "a"] |

1. Trả Về 1 Phần Tử Trong List Và Xóa Nó?

<Phần Tử> = <List>.pop(<Index>)

* Nếu <Index> không được chỉ định thì sẽ chọn phần tử cuối
* Ví dụ

foo = bar.pop(1)

* Ta có

|  |  |
| --- | --- |
| bar trước | [4, 2, 9, 8, 7] |
| foo | 2 |
| bar sau | [4, 9, 8, 7] |

1. Trả Về Index Của Phần Tử Đầu Tiên Có Giá Trị Nào Đó?

<Index> = <List>.index(<Giá Trị>)

* Ví dụ

foo = bar.index(4)

* Ta có

|  |  |
| --- | --- |
| bar | [1, 2, 4, 5, 6, 4, 2] |
| foo | 2 |

1. Cách Tạo 1 List Trống Với Độ Dài Bất Kì?

* Để tạo 1 List gồm các số với cùng 1 giá trị

[<Giá Trị>] \* <Độ Dài>

* Các phần tử trong List trả về trong trường hợp này không tham chiếu đến nhau
* Ví dụ

foo = [8] \* 4

* Ta có

|  |  |
| --- | --- |
| foo | [8, 8, 8, 8] |

Generator:

1. Cách Trả Về Phần Tử Tiếp Theo Của Generator Và Xóa Nỏ Khỏi Generator?

next(<Generator>)

Set – Tập Hợp:

1. Set Giải Thích Dễ Hiểu?

* Giống như tập hợp bình thường trong toán nghĩa là không quan tâm thứ tự, mỗi phần tử là độc nhất

1. Tạo Nhanh 1 Set?

<Biến> = {<Các giá trị>}

* Ví dụ

foo = {1, 2, “bar”}

Enumerate – Liệt Kê:

1. Enumerate Giải Thích Dễ Hiểu?

* Giống như liệt kê bình thường, đánh số theo thứ tự 0, 1, 2, … ứng với các giá trị nào đó

1. Tạo 1 Enumerate?

<Biến> = enumerate(<Mảng>, <Start>)

* Mỗi phần tử trong <Biến> sẽ có 2 giá trị, đầu tiên là số thứ tự, thứ 2 là giá trị tương ứng trong mảng
* Số thứ tự đầu tiên là <Start>
* Ví dụ

foo = enumerate([5, ‘a’, 4], 2)

* foo giống kiểu {2 : 5, 3 : ‘a’, 4 : 4}

Dictionary – Từ Điển:

1. Dictionary Giải Thích?

* Giống như từ điển bình thường, bạn tra từ để tìm giải thích của nó, giống Set, nhưng mỗi phần tử gồm 2 thằng là Key và giá trị, dùng Key để truy xuất giá trị

1. Tạo Nhanh 1 Dictionary?

<Biến> = {<Các Cặp Key Và Giá Trị>}

* Key có thể là bất kì kiểu dữ liệu nào ngoại trừ List
* Ví dụ

foo = {“cac” : 1, 2 : “bar”, (1, 2) : [4, 5]}

* foo[(1, 2)] trả về [4, 5]

1. Trả Về Số Key Của Dictionary?

<Số Key> = len(<Dictionary>)

1. Chuyển Kiểu Dữ Liệu Khác Về Dictionary?

* Giả sử “foo” là 1 đối tượng Iterable, chứa các phần tử cũng Iterable, các Iterable con này đều chứa 2 phần tử, nhập lệnh sau

<Dictionary> = dict(foo)

* <Dictionary> sẽ có các cặp Key và giá trị tương ứng với 2 phần tử trong các Iterable con
* Ví dụ

foo = ([1, ‘a’], (‘abc’, 5))

bar = dict(foo)

* “bar” có giá trị là {1 : “a”, “abc” : 5}

1. Xóa 1 Phần Tử Trong Dictionary Và Trả Về Giá Trị Tương Ứng?

<Giá Trị> = <Dictionary>.pop(<Key>)

* Ví dụ

foo = bar.pop("far")

* Ta có

|  |  |
| --- | --- |
| bar trước | {"far" : 123, 4 : 9, 5 : 7} |
| bar sau | {4 : 9, 5 : 7} |
| foo | 123 |

1. Trả Về Giá Trị Tương Ứng Với 1 Key Cho Dù Key Đó Chưa Tồn Tại?

<Giá Trị 1> = <Dictionary>.setdefault(<Key>, <Giá Trị 2>)

* Nếu <Key> đã tồn tại trong <Dictionary>, lệnh trên đéo làm gì, trừ việc gán giá trị tương ứng với <Key> cho <Giá Trị 1>
* Nếu <Key> chưa tồn tại, thì nó sẽ được thêm vào <Dictionary> ứng với <Giá Trị 2>, đồng thời <Giá Trị 1> = <Giá Trị 2>
* Ví dụ

foo = bar.setdefault(4, 10)

boo = far.setdefault(1, 8)

* Ta có

|  |  |
| --- | --- |
| bar trước | {"far" : 123, 4 : 9, 5 : 7} |
| bar sau | {"far" : 123, 4 : 9, 5 : 7} |
| foo | 9 |
| far trước | {"far" : 123, 4 : 9, 5 : 7} |
| far sau | {"far" : 123, 4 : 9, 5 : 7, 1 : 8} |
| boo | 8 |

1. Lặp Qua Dictionary?

* Lặp qua Key

for <Key> in <Dictionary>:

<Làm Gì Đó>

* Lặp qua Value

for <Value> in <Dictionary>.values()

<Làm Gì Đó>

Class – Lớp:

1. Bản Chất Class?

* Tưởng tượng Class giống như đường dẫn đến File, giả sử Class to nhất của bạn là A, thì A là thư mục gốc, tạo Class B bên trong A, tương đương tạo thư mục con B trong thư mục gốc A, tạo thuộc tính và hàm trong A, tương đương tạo File trong thư mục A
* Để truy cập 1 thuộc tính hoặc hàm, thì phải biết đường dẫn đến nó, khác với đường dẫn thông thường, đường dẫn này thay dấu “/” thành “.”
* Ví dụ
* Class A to nhất, trong đó có Class B, trong B có thuộc tính “foo” = 5, khi đó “A.B.foo” có giá trị = 5, trong B còn có hàm “bar”, khi đó “A.B.bar(<Tham Số>)” sẽ gọi hàm này

1. Bản Chất Của 1 Đối Tượng Thuộc 1 Class Nào Đó?

* Để tạo 1 đối tượng của 1 Class, ví dụ tạo đối tượng B từ Class A, viết

B = A()

* Đầu tiên, xét thuộc tính, giả sử ta có Class A, trong đó có “foo” = 4, B là 1 đối tượng thuộc Class A, truy cập tất cả thuộc tính của Class A = đối tượng B như bình thường, tuy nhiên “foo” dùng chung, nghĩa là nếu thay đổi “foo” thông qua “A.foo” thì “B.foo” cũng thay đổi tương tự, nhưng nếu dùng “B.foo” để thay đổi, thì “A.foo” và tất cả các đối tượng của class A sẽ không thay đổi, khi này, “foo” của B sẽ tách ra làm thuộc tính riêng của B
* Xét hàm số, trong Class A có hàm “bar”, khi này dùng B để gọi “bar” sẽ có 1 chút khác, đó là tham số đầu tiên của “bar” sẽ chính là bản thân B luôn, nên bạn có thể dùng tham số này để thay đổi hàm và thuộc tính của B, thông thường nó được đặt tên là “self”
* Nếu bạn muốn không có sự khác biệt, nghĩa là không có tham số đầu tiên con mẹ gì hết, thì đặt @staticmethod lên trước khai báo hàm
* Nếu bạn muốn khi dùng B để gọi “bar”, và muốn tham số đầu tiên của “bar” thay vì là chính bản thân B, thì nó lại chính là Class A luôn, thì đặt @classmethod lên trước khai báo hàm
* Hàm “\_\_init\_\_” là trường hợp đặc biệt, sau khi B đăng kí Class A, nó sẽ ngay lập tức gọi hàm này
* Hàm “\_\_getitem\_\_” cũng đặc biệt, nếu bạn viết “B[x, y, …]” thì nó sẽ tự động gọi hàm này và Pass Tuple (x, y, …) vào
* Hàm “\_\_len\_\_” cũng đặc biệt, nếu bạn viết “len(B)” thì nó sẽ tự động gọi hàm này

1. Tạo Nhanh 1 Class?

class <Tên Class>(<Tên Class Kế Thừa>):

def \_\_init\_\_(self, <Tham Số Khởi Tạo>):

super().\_\_init\_\_(<Giá Trị Khởi Tạo Class Kế Thừa>)

self.<Tên Thuộc Tính> = <Giá Trị Thuộc Tính>

def <Tên Hàm Trong Class>(self, <Tham Số>):

<Xử Lí Gì Đó Với Tham Số>

return <Giá Trị Gì Đó>

* Lệnh “super().\_\_init\_\_()” dùng để khởi tạo Class kế thừa với những giá trị mình đưa vào
* Ví dụ

class foo(bar):

def \_\_init\_\_(self, bruh):

super().\_\_init\_\_()

self.dark = “lmao”

def lol(self, shit):

return shit + 1

new\_foo = foo(4)

1. Check 1 Object Xem Nó Thuộc Class Nào?

<Có Thuộc Class Này Không> = isinstance(<Object>, <Class>)

* Ví dụ

foo = isinstance((2, 3), tuple)

* Ta có

|  |  |
| --- | --- |
| foo | True |

1. Check Xem 1 Object Hoặc Class Có Thuộc Tính Nào Đó Không?

<Có Thuộc Tính Này Không> = hasattr(<Object Hoặc Class>, <Tên Thuộc Tính>)

* Ví dụ

foo = hasattr(foo, "bar")

* Ta có

|  |  |
| --- | --- |
| foo | True |

Error – Lỗi:

1. Cách Nhanh Nhất Để Thông Báo Lỗi Nếu 1 Biểu Thức Logic Nào Đó False?

assert <Biểu Thức Logic>

* ví dụ

assert 1 > 5

* Màn hình sẽ thông báo “Assertion Error”

assert 1 < 5

* Câu lệnh này được bỏ qua

Function – Hàm:

1. Positional Argument Và Key Word Argument?

* Xét ví dụ sau

foo = bar(1, "a", [5, 6], shit = "bob", boo = 2, fool = [4, "aa"])

* Ta có

|  |  |
| --- | --- |
| Positional Argument | 1, "a", [5, 6] |
| Keyword Argument | shit = "bob", boo = 2, fool = [4, "aa"] |

* Positional Argument không được phép đứng sau Keyword Argument

1. Default Argument Và Non Default Argument?

* Xét ví dụ sau

def foo(bar, boo, far = 5, fool = "a")

* Ta có

|  |  |
| --- | --- |
| Default Argument | far = 5, fool = "a" |
| Non Default Argument | bar, boo |

* Khi gọi hàm “foo”, ta không nhất thiết phải Pass giá trị cho “far” và “fool”
* Ví dụ

foo(2, 3, 9)

* Ta có

|  |  |  |  |
| --- | --- | --- | --- |
| bar | 2 | boo | 3 |
| far | 9 | fool | "a" |

1. Cơ Chế Hoạt Động Của “\*” Và “\*\*” Ở Tham Số Hàm?

* Theo thường lệ, thì ta sẽ đặt tham số hàm theo thứ tự là Non Default Argument, Default Argument, “\*”, “\*\*”
* Xét ví dụ sau

def foo(bar, boo, alice = 5, bob = 7, \*far, \*\*fool)

* Khi Pass giá trị cho “foo”, ta có

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| foo(1, 2, 3, dark = 4, bob = 8, shit = 7) | bar | 1 | bob | 8 |
| boo | 2 | far | () |
| alice | 3 | fool | {"dark" : 4, "shit" : 7} |
| foo(1, 2, 3, 4, 5, 6, 7, 8, 9, light = 10) | bar | 1 | bob | 4 |
| boo | 2 | far | (5, 6, 7, 8, 9) |
| alice | 3 | fool | {"light" : 10} |

1. Cách Sử Dụng “\*” Để Unpacking?

* Khi Unpacking, chỉ có 1 biến được chứa “\*”, ví dụ

foo, \*bar, boo, ha = [1, 4, “far”, “bob”, 5, 9]

* “foo” có giá trị là 1, “bar” có giá trị là [4, “far”, “bob”], “boo” có giá trị 5 và “ha” có giá trị 9

Quick Code – Mã Nhanh:

1. Chỉnh Tên Của Tất Cả Các Ảnh PNG Trong 1 Thư Mục Thành Số Thự Tự Tăng Dần?

* Áp dụng cho tập ảnh bị thiếu số thứ tự cần Offset

import os

FOLDER\_PATH = r<Đường Dẫn Tới Thư Mục>

FILE\_EXTENSION = '.png'

file\_list = [f for f in os.listdir(FOLDER\_PATH) if f.endswith(FILE\_EXTENSION)]

file\_list.sort(key = lambda x: int(os.path.splitext(x)[0]))

for index, old\_name in enumerate(file\_list, 1):

new\_name = f"{index}.png"

old\_path = os.path.join(FOLDER\_PATH, old\_name)

new\_path = os.path.join(FOLDER\_PATH, new\_name)

os.rename(old\_path, new\_path)

1. Xóa Toàn Bộ Comment Trong 1 Python File?

import re

FILE\_PATH = <Đường Dẫn Tới File>

with open(FILE\_PATH, "r") as f:

content = f.read()

content = re.sub(r'#.\*', '', content)

content = re.sub(r"'''(.\*?)'''", '', content, flags=re.DOTALL)

with open(FILE\_PATH, 'w') as f:

f.write(content)

1. Xóa Toàn Bộ Doc String Trong 1 Python File?

import re

FILE\_PATH = <Đường Dẫn Tới File>

with open(FILE\_PATH, "r") as file:

code = file.read()

pattern = r'r""".\*?"""'

cleaned\_code = re.sub(pattern, '', code, flags=re.DOTALL)

with open(FILE\_PATH, "w") as file:

file.write(cleaned\_code)

1. Xóa Toàn Bộ Dòng Trống Trong 1 Python File?

FILE\_PATH = <Đường Dẫn Tới File>

with open(FILE\_PATH, "r") as file:

lines = file.readlines()

non\_empty\_lines = [line for line in lines if line.strip() != ""]

with open(FILE\_PATH, "w") as file:

file.writelines(non\_empty\_lines)

1. Đổi Phần Mở Rộng Của Tất Cả Các File Trong 1 Thư Mục Thành Phần Mở Rộng Nào Đó?

import os

FOLDER\_PATH = <Đường Dẫn Tới Thư Mục>

NEW\_EXTENSION = <Phần Mở Rộng Mới>

FILE\_LIST = os.listdir(FOLDER\_PATH)

for filename in FILE\_LIST:

if os.path.isfile(os.path.join(FOLDER\_PATH, filename)):

name, ext = os.path.splitext(filename)

if ext != NEW\_EXTENSION:

new\_filename = name + NEW\_EXTENSION

old\_path = os.path.join(FOLDER\_PATH, filename)

new\_path = os.path.join(FOLDER\_PATH, new\_filename)

os.rename(old\_path, new\_path)

* Ví dụ

|  |  |
| --- | --- |
| <Đường Dẫn Tới Thư Mục> | "./bar" |
| <Phần Mở Rộng Mới> | ".jpg" |

Operator – Toán Tử:

1. Số Phức?

* Tạo số phức

<Số Phức> = <Phần Thực> + <Phần Ảo>j

* Ví dụ

foo = 2 + 3j

* Để trả về phần thực

<Số Phức>.real

* Để trả về phần ảo

<Số Phức>.imag

1. Hợp 2 Tập Hợp?

<Hợp> = <Tập 1> | <Tập 2>

* Giống y chang trong toán
* <Tập 1> và <Tập 2> phải là Set
* Ví dụ

foo = {1, 2, 3} | {2, 5, 9}

* Ta có

|  |  |
| --- | --- |
| foo | {1, 2, ,3, 5, 9} |

1. Làm Tròn Số?

<Số Làm Tròn> = round(<Số Gốc>, <Số Chữ Số Sau Dấu Phẩy>)

* Nếu <Số Chữ Số Sau Dấu Phẩy> âm thì làm tròn qua cả phần nguyên
* Ví dụ

foo = round(bar, -2)

* Ta có

|  |  |
| --- | --- |
| bar | 1234.5678 |
| foo | 1200 |