



KỸ THUẬT

LẬP TRÌNH

M'

01

DỮ LIỆU

Cấu trúc dữ liệu,
chuyển đổi,
ứng dụng

02

KỸ THUẬT

Đọc, ghi, xử lý,
lưu trữ dữ liệu
Lệnh điều kiện,
vòng lặp

03

HÀM

Định nghĩa, khởi tạo,
cách sử dụng, lưu ý,
ứng dụng



04

ĐỆ QUY

Định nghĩa, ứng dụng, khử đệ quy

05

KIỂM THỬ

Thử nghiệm, đọc code, dò lỗi, gỡ lỗi, kiểm soát lỗi

06

PHONG CÁCH

Quy ước đặt tên, bố trí dòng lệnh



01

DỮ LIỆU

Cấu trúc dữ liệu, chuyển đổi,
ứng dụng



PHÂN LOẠI DỮ LIỆU TỰ NHIÊN VÀ CHUYỂN ĐỔI

Loài người phát triển nhờ trao đổi thông tin để phối hợp ăn ý với nhau.

Để máy tính làm việc chung với con người, ta cần phân loại và chuyển đổi qua lại giữa dữ liệu tự nhiên và dữ liệu máy

3 NHÓM DỮ LIỆU CHÍNH



LOGIC

Kiểm tra tính đúng sai, hỗ trợ đưa ra quyết định



SỐ HỌC

Thể hiện độ lớn giá trị trong thực tiễn, có thể đo lường tính toán bằng đơn vị



DANH SÁCH

Tập hợp các giá trị riêng lẻ được nhóm lại với nhau theo trật tự, có thể dùng tách biệt từng giá trị

3 NHÓM DỮ LIỆU CHÍNH



LOGIC

bool



SỐ HỌC

int, float, complex



DANH SÁCH

str, list, tuple, range,
dict, set

ĐÁNH GIÁ
HỌC SINH?

PASS FAIL

Đạt chuẩn

0 1 2 3 4 5 6 7 8 9 10

Thang điểm

T Toán:	<input type="text" value="0.0"/>	V Vật lý:	<input type="text" value="0.0"/>	H Hóa học:	<input type="text" value="0.0"/>
S Sinh học:	<input type="text" value="0.0"/>	N Ngữ văn:	<input type="text" value="0.0"/>	L Lịch sử:	<input type="text" value="0.0"/>
Đ Địa lý:	<input type="text" value="0.0"/>	N Ngoại ngữ:	<input type="text" value="0.0"/>	G GD&CD:	<input type="text" value="0.0"/>

Bảng điểm

Những kiểu dữ liệu đơn giản

```
1 # Kiểu bool: True hoặc False
2 isValid = True                    # True
3
4 # Kiểu int: Số nguyên
5 page = 24                        # 24
6
7 # Kiểu float: Số thực
8 weight = 3.5                     # 3.5
9
10 # Kiểu complex: Số phức
11 pendulum = 3 + 2j                # (3+2j)
12
13 # Kiểu str: Chuỗi ký tự
14 quote = 'No pain no gain.'      # No pain no gain.
15
16 # Kiểu range: dải số
17 scale = range(3, 7, 2)           # range(3, 7, 2) # dãy số: 3, 5
```

Những danh sách đặc biệt

Cấu trúc	Thứ tự	Bất biến	Dữ liệu duy nhất	Ứng dụng chính
Tuple	Có	Có	Không	Lưu trữ dữ liệu không thay đổi, trả về nhiều giá trị cùng lúc
Dictionary	Có	Không	Khóa	Dùng các cặp khóa-giá trị để lưu trữ và truy xuất bằng khóa
Set	Không	Không	Tất cả	Lọc dữ liệu duy nhất, toán học tập hợp
List	Có	Không	Không	Được sử dụng nhiều nhất, hầu hết với dữ liệu có thứ tự

Những danh sách đặc biệt

```
1 # Kiểu tuple
2 location = (21.4912, 103.0023)
3 # (21.4912, 103.0023)
4
5 # Kiểu dict
6 person = {'name': 'Duy', 'age': 28, 'job': 'teacher'}
7 # {'name': 'Duy', 'age': 28, 'job': 'teacher'}
8
9 # Kiểu set
10 hobbies = {'reading', 'traveling', 'sports', 'music', 'sports'}
11 # {'reading', 'traveling', 'sports', 'music'}
12
13 # Kiểu list
14 shopping_list = ['milk', 'eggs', 'bread', 'butter']
15 # ['milk', 'eggs', 'bread', 'butter']
```

Ứng dụng thực tế

Bạn sẽ mã hóa những thông tin này như thế nào cho máy tính hiểu được?

- Bạn xem trời có mưa không để quyết định có mang theo ô đi ra ngoài không?
- Thủ thư cần liệt kê số lượng sách đã cho mượn và tỷ lệ hoàn trả đúng hạn
- Ban quản lý chung cư cần thống kê mức sử dụng điện nước của từng hộ cư dân để gửi hóa đơn thanh toán tiền

Ứng dụng thực tế

Bạn sẽ mã hóa những thông tin này như thế nào cho máy tính hiểu được?

- Bạn xem trời có mưa không để quyết định có mang theo ô đi ra ngoài không?
Nhóm logic: `bool`
- Thủ thư cần liệt kê số lượng sách đã cho mượn và tỷ lệ hoàn trả đúng hạn
Nhóm số học: `int, float`
- Ban quản lý chung cư cần thống kê mức sử dụng điện nước của từng hộ cư dân để gửi hóa đơn thanh toán tiền
Nhóm danh sách: `list, dict`



02

KỸ THUẬT

Đọc, ghi, xử lý, lưu trữ dữ liệu,
lệnh điều kiện, vòng lặp



NHỮNG KỸ THUẬT CƠ BẢN QUAN TRỌNG

Sau đây là những kỹ thuật căn bản nhưng quan trọng, đặt nền móng cho sự nghiệp IT

BIẾN

Thứ có thể lưu trữ các giá trị tạm thời và có thể thay đổi được

LƯU TRỮ TẠM THỜI:

- Có thể mang bất kỳ kiểu dữ liệu nào

CÓ THỂ THAY ĐỔI:

- Giá trị của biến không cố định, có thể bị thay đổi bằng lệnh gán



Ví dụ biến thức tế

Cũng là in ra biến a
nhưng kết quả thì
khác nhau, phụ
thuộc hoàn toàn vào
lệnh gán trước đó

```
1 a = 10
2 print(a)
3 a = True
4 print(a)
5 a = 'Coding'
6 print(a)
7 a = [1, 9, 4, 5]
8 print(a)
```

```
10
True
Coding
[1, 9, 4, 5]
```

```
** Process exited - Return Code: 0 **
Press Enter to exit terminal
```

TOÁN TỬ

Các phép tính thông dụng xử lý tương tác giữa các dữ liệu với nhau

SỐ HỌC:

- + - * /
- % // **

GÁN:

- =
- += -= *= /=
- %= //= **=

LOGIC:

- == > < != >= <=
- and or not

Đại số Boole

NOT

- `not True == False`
- `not False == True`

AND

- `True and True == True`
- `True and False == False`
- `False and True == False`
- `False and False == False`

OR

- `True or True == True`
- `True or False == True`
- `False or True == True`
- `False or False == False`

Ví dụ toán tử số học

Thứ tự ưu tiên sẽ là:

- Trong ()
- **
- * / % //
- + -
- Trái sang phải

Đáp án:

- **a = 29.0**

```
1 # Dự đoán kết quả của phép tính sau
2
3
4 a = 30 + 38 / 2 // 3 ** 2 * 3 % 5 - 2
5
6
7 print(a)
```


Đặc biệt

Toán tử + và * có thể xử lý chuỗi:

- + : Nối chuỗi
- * : Lặp chuỗi

Đáp án:

- **a = 33**
- **b = [3, 3, 3]**

```
1 # Dự đoán kết quả của phép tính sau
2
3
4 a = '3' + '3'
5 b = 3 * [3]
6
7
8 print(a)
9 print(b)
```

Ví dụ toán tử logic

- Toán tử logic hỗ trợ xử lý những điều kiện phức tạp, kết quả trả về luôn là kiểu logic đúng sai

Đáp án:

- **choice = False**

```
1  '''
2  Một quán ăn phục vụ theo món ăn mà khách hàng bốc thăm được
3  Quán sử dụng sợi bún và bánh đa
4  Món ăn kèm gồm: cá, hải sản và thập cẩm
5
6  Giả sử An muốn ăn bún hải sản nhưng bốc thăm ra bún cá
7  '''
8
9  noodles = 'bún'
10 #noodles = 'bánh đa'
11
12 topping = "cá"
13 #topping = "hải sản"
14 #topping = "thập cẩm"
15
16 choice = noodles == 'bún' and topping == "hải sản"
17
18 print(choice)
```

Ví dụ toán tử gán

Việc dùng toán tử gán sẽ thay đổi giá trị của biến thành giá trị mới

Đáp án:

- **a = 4.0**

```
1 # Dự đoán kết quả của a
2
3 a = 32
4 a /= 2
5 a /= 2
6 a /= 2
7
8
9 print(a)
```

TRUY XUẤT DỮ LIỆU

```
1 # Tính hóa đơn mua hàng
2
3 prices = [120000, 2500, 12000, 18000]
4 # Theo thứ tự là giá thịt lợn(kg), đậu phụ(miếng), cà chua(kg) và hành lá(kg)
5
6 pork, tofu, tomato, scallion = 0.3, 4, 0.5, 0.1
7
8 payment = pork * prices[0] + tofu * prices[1] + tomato * prices[2] + scallion * prices[3]
9
10 print(int(payment))
```

53800

** Process exited - Return Code: 0 **
Press Enter to exit terminal

Dữ liệu **điều kiện** và **số học** có thể lấy trực tiếp

Dữ liệu **danh sách** cần dùng thêm **chỉ số** hoặc **khóa** để hỗ trợ truy xuất

Lệnh truy xuất **không làm thay đổi** giá trị của biến

String, List và Tuple

```
1 # Cách truy xuất dữ liệu của str, list và tuple là tương tự nhau
2
3 # a = 'bright'
4 # a = ['Mon', 'Tue', 'Wed', 'Thu', 'Fri', 'Sat', 'Sun']
5 a = (1, 2, 3, 4, 5, 6, 7)
6
7 # res = a[1]      # Lấy phần tử theo chỉ số
8 # res = a[-2]     # Lấy phần tử theo chỉ số ngược
9 # res = a[3: 5]   # Lấy các phần tử từ 3 đến 4
10 # res = a[:2]     # Lấy các phần tử từ đầu đến 1
11 # res = a[3:]     # Lấy các phần tử từ 3 đến cuối
12
13 # Sắp xếp theo list tăng dần
14 # res = sorted(a)
15
16 # Sắp xếp theo list giảm dần
17 res = sorted(a, reverse = True)
18
19 print(res)
```

Cách truy xuất dữ liệu của str, list và tuple về cơ bản là tương tự nhau.

Mấu chốt nằm ở việc chúng đều có biến chỉ số(index) chỉ chính xác vị trí của dữ liệu

Dictionary

```
1 ''' Kiểu dict là công cụ rất đặc lực giúp mã hóa những danh sách
2 phức tạp theo cách hiểu của của con người thông qua cặp key-value'''
3
4 person = {'name': 'Duy', 'age': 28, 'job': 'teacher'}
5
6 # Lấy value ứng với key nhưng sẽ lỗi KeyError nếu key không tồn tại
7 res = person['name']
8
9 # Cũng lấy value ứng với key nhưng trả về None nếu không tồn tại key
10 # thay vì báo lỗi KeyError
11 res = person.get('gender')
12
13 # Trả về danh sách các key ở dạng list
14 res = person.keys()
15
16 # Trả về danh sách các value ở dạng list
17 res = person.values()
18
19 print(res)
```

Khác với các danh sách trước, dict sử dụng khóa(**key**) để đánh dấu dữ liệu.

Bản chất của **key** và **index** là giống nhau, đều để đánh dấu dữ liệu, tuy nhiên **key** thể hiện được rõ ràng ý nghĩa của dữ liệu

CẬP NHẬT DỮ LIỆU

```
1 board = [1, 9, 4, 5]
2
3 # Thay đổi giá trị theo chỉ số trong danh sách
4 board[2] = 7          # board = [1, 9, 7, 5]
5
6 # Thêm phần tử vào cuối danh sách
7 board.append(4)       # board = [1, 9, 7, 5, 4]
8
9 # Loại bỏ phần tử theo chỉ số
10 remove = board.pop(4) # remove = 4
11          # board = [1, 9, 7, 5]
12
13 # Chèn thêm 1 phần tử
14 board.insert(0, 4)    # board = [4, 1, 9, 7, 5]
15
16 # Thay đổi 1 phần đoạn trong danh sách
17 board[0:1] = [3, 0, 4] # board = [3, 0, 4, 1, 9, 7, 5]
18
19 print(board)
```

List và Dict là các công cụ hữu dụng để ghi chép các thay đổi trong việc xử lý dữ liệu trước khi trả ra kết quả. Lệnh cập nhật sẽ **làm thay đổi** giá trị của biến.

CẬP NHẬT DỮ LIỆU

```
1 monster1 = {'name': 'Dark Magician Girl', 'attack': 2000, 'defend': 1700}
2
3 # Thay đổi value ứng với key
4 monster1['attack'] = 2300
5 # {'name': 'Dark Magician Girl', 'attack': 2300, 'defend': 1700}
6
7 # Thêm một cặp key-value
8 monster1['attribute'] = 'Dark'
9 # {'name': 'Dark Magician Girl', 'attack': 2300, 'defend': 1700, 'attribute': 'Dark'}
10
11 # Thay đổi nhiều value hoặc thêm nhiều cặp key-value
12 monster1.update(attack=2600, defend=1500, level=6)
13 # {'name': 'Dark Magician Girl', 'attack': 2600, 'defend': 1500, 'attribute': 'Dark', 'level': 6}
14
15 print(monster1)
```

List và Dict là các công cụ hữu dụng để ghi chép các thay đổi trong việc xử lý dữ liệu trước khi trả ra kết quả. Lệnh cập nhật sẽ **làm thay đổi** giá trị của biến.

LỆNH ĐIỀU KIỆN

Khi chương trình cần phân chia thành nhiều kế hoạch hành động khác nhau, ta cần lệnh điều kiện để lựa chọn

IF điều kiện:

- Hành động khi điều kiện đúng

ELSE:

- Hành động khi điều kiện sai

Ứng dụng lệnh điều kiện

```
1 # Bài toán bán vé xem xiếc theo độ tuổi
2 # Dưới 6 tuổi - Miễn phí
3 # Từ 6 đến dưới 18 tuổi - 50k
4 # Từ 18 tuổi trở lên - 100k
5
6 age = 38
7
8 if age < 6:
9     ticketPrice = 0
10 elif age >= 18:
11     ticketPrice = 100000
12 else:
13     ticketPrice = 50000
14
15 print(f'Bạn ở độ tuổi {age}. Giá vé của bạn là {ticketPrice}đ.')
```

Các điều kiện ở elif và else phải có liên quan với điều kiện ở if và không có phần trùng lặp với điều kiện ở trước nó.

Nếu có nhiều điều kiện phức tạp không liên quan đến nhau thì nên dùng lệnh điều kiện mới

VÒNG LẶP

Khi một việc cần thực hiện tuần tự nhiều lần mới cho ra kết quả, chúng ta cần đến vòng lặp

WHILE:

- Số lần lặp không xác định cụ thể
- Hoạt động xoay quanh điều kiện dừng

FOR:

- Số lần lặp xác định rõ ràng
- Hoạt động xoay quanh biến chỉ số (**index**)



Vòng lặp while

```
1 # Đăng ký tài khoản mới với tên đăng nhập là dạng chữ
2 # và mật khẩu là dạng số, không chứa ký tự đặc biệt
3
4 username = ''
5 password = ''
6
7 while not username.isalpha():
8     username = input('Username: ')
9 while not(password.isdigit() and len(password) == 6):
10     password = input('Password: ')
11
12 print('\nHello ' + username + '!')
```

ƯU ĐIỂM:

- Có thể dùng linh hoạt trong mọi trường hợp

NHƯỢC ĐIỂM:

- Không có sẵn biến chỉ số, khó kiểm soát số lần lặp
- Gây lỗi lặp vô hạn khi không kiểm soát tốt điều kiện dừng

Vòng lặp for

```
1 # Ngân hàng chỉ cho thử đăng nhập 5 lần
2 # để bảo mật tài khoản khách hàng
3
4 accounts = [
5     {'username': 'alex', 'password': '300475'},
6     {'username': 'david', 'password': '190845'},
7     {'username': 'emily', 'password': '270173'}
8 ]
9
10 user = 'emily'
11 print('Username: ' + user)
12
13 for i in range(5):
14     isValid = False
15     password = input('Password: ')
16     for acc in accounts:
17         if acc['username'] == user and acc['password'] == password:
18             isValid = True
19             break
20     if isValid == True:
21         print('Hello ' + user + '!')
22         break
```

ƯU ĐIỂM:

- Có sẵn biến chỉ số tiện dụng trong nhiều trường hợp

NHƯỢC ĐIỂM:

- Thiếu linh hoạt vì số lần lặp bị cố định từ trước
- Khó xử lý với điều kiện lặp phức tạp

Các lệnh ngắt vòng lặp

BREAK

- Ngắt toàn bộ vòng lặp
- Sử dụng khi vòng lặp đã hoàn thành nhiệm vụ của mình và không cần chạy nốt các bước lặp còn lại

CONTINUE

- Ngắt 1 bước trong vòng lặp
- Sử dụng để bỏ qua một bước không còn cần thiết nữa nhưng vòng lặp vẫn chưa hoàn thành nhiệm vụ và vẫn phải chạy tiếp các bước lặp còn lại

Ví dụ lệnh ngắt vòng lặp

- Cường muốn gửi email cho các bạn nam trong lớp để bàn bạc chuyện tổ chức 8/3 cho các bạn nữ

Đáp án:

- ['binh@em.com',
'kien@em.com']**

```
1 students = [{"name": "An", "gender": "Female", "email": "an@em.com"},
2             {"name": "Bình", "gender": "Male", "email": "binh@em.com"},
3             {"name": "Cường", "gender": "Male", "email": "cuong@em.com"},
4             {"name": "Giang", "gender": "Female", "email": "giang@em.com"},
5             {"name": "Hương", "gender": "Female", "email": "huong@em.com"},
6             {"name": "Kiên", "gender": "Male", "email": "kien@em.com"},
7             {"name": "Lan", "gender": "Female", "email": "lan@em.com"},
8             {"name": "Mai", "gender": "Female", "email": "mai@em.com"}]
9
10 emails = []
11
12 for student in students:
13     if student["name"] == "Cường":
14         continue
15     if student["gender"] == "Male":
16         emails.append(student["email"])
17     if len(emails) == 2:
18         break
19
20 print(emails)
```


CHANGE YOUR MIND

Khi chúng ta nhìn đúng góc độ,
mọi thứ trở nên đơn giản hơn rất nhiều





03

HÀM

Định nghĩa, khởi tạo, cách sử dụng, lưu ý, ứng dụng

**MỖI BỘ
PHẬN TRÊN
CƠ THỂ CÓ
CHỨC NĂNG
RIÊNG**



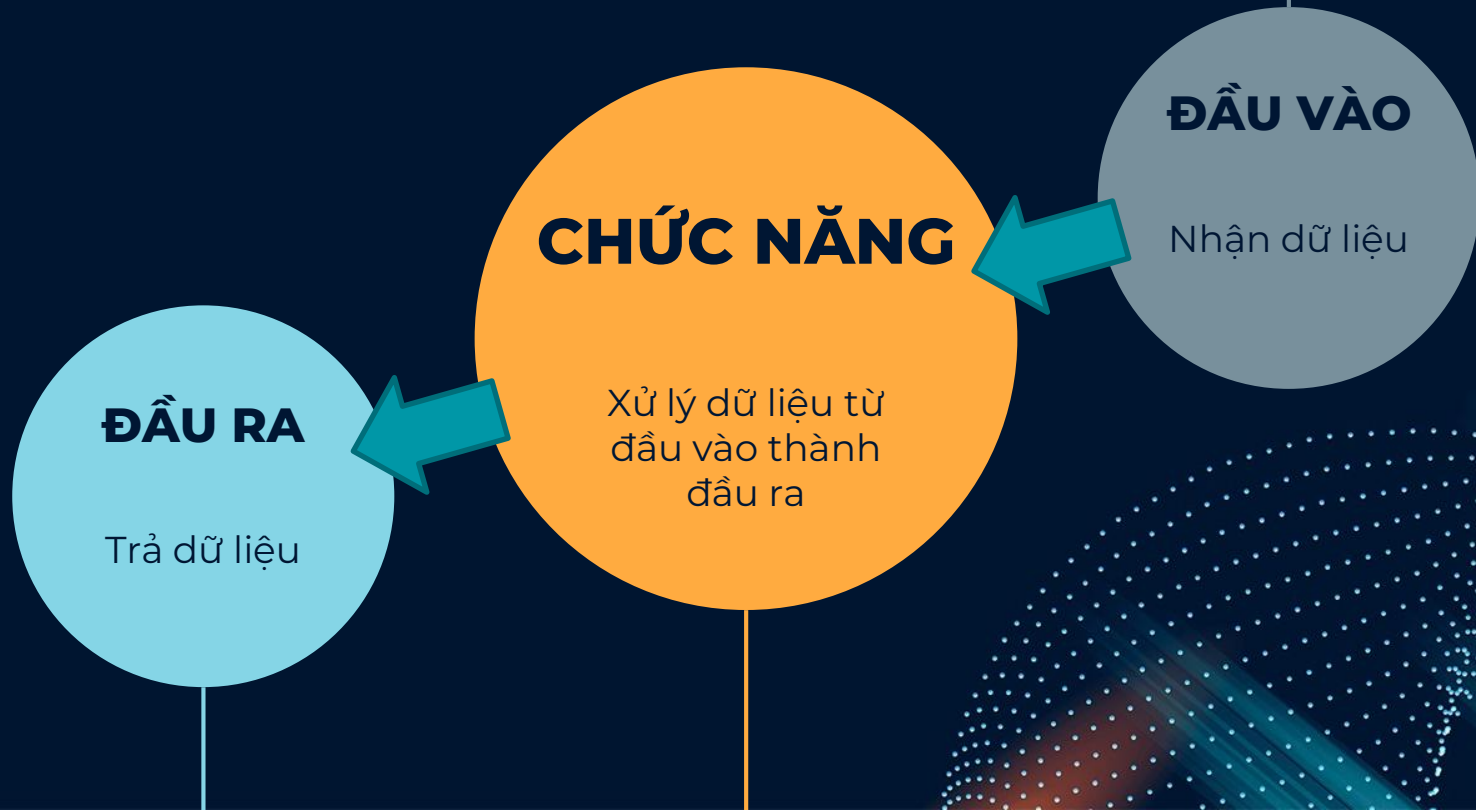
VÀ PROJECT CŨNG ĐƯỢC TẠO NÊN TỪ CÁC HÀM LÀ MẢNH GHÉP...



● Factor 1 ● Factor 2
● Factor 3 ● Factor 4



CẤU TẠO CỦA MỘT HÀM



Ví dụ hàm thực tế

Luôn đảm bảo đầy đủ cấu tạo cho hàm để chúng làm tốt nhiệm vụ của mình. Kiểm soát đầu vào, đầu ra để tránh lỗi.

```
1 def pythagoras(edge1, edge2):  
2     hypotenuse = (edge1**2 + edge2**2)**0.5  
3     return hypotenuse  
4  
5 print(pythagoras(3, 4))
```

Tham số mặc định

```
1 ''' Một quán cà phê hầu gái tuyển nhân viên làm việc theo ngày lẻ
2 với thời gian làm việc và lương theo thỏa thuận riêng
3 hoặc với ca mặc định là 8h, lương 50k/giờ.'''
4
5 def day_salary(work_hours=8, hourly_rate=50):
6     salary = work_hours * hourly_rate
7     return salary
8
9 # Eimi chỉ muốn làm 5h rồi đi học
10 print(f'Eimi: {day_salary(5)}k')
11
12 # Yua nổi tiếng nên thỏa thuận lương 100k/h
13 print(f'Yua: {day_salary(hourly_rate=100)}k')
14
15 # Arina làm theo ca và lương mặc định
16 print(f'Arina: {day_salary()}k')
17
18 # Minami tăng ca làm 10h và được tăng lương lên 60k/h
19 print(f'Minami: {day_salary(10, 60)}k')
```

Khi không truyền giá trị đầu vào cho tham số mặc định, nó sẽ **tự lấy giá trị được gán** làm giá trị đầu vào

PHỐI HỢP CÁC HÀM

```
1 # Hàm tính cạnh huyền bằng Pythagoras
2 def pythagoras(edge1, edge2):
3     hypotenuse = (edge1**2 + edge2**2)**0.5
4     return hypotenuse
5
6 # Hàm tính đường cao ứng với cạnh huyền
7 def altitude(edge1, edge2):
8     # Sử dụng hàm pythagoras() để tính cạnh huyền
9     hypotenuse = pythagoras(edge1, edge2)
10
11     altitude = edge1 * edge2 / hypotenuse
12     return altitude
13
14 altitude = altitude(6.2, 8)
15 altitude = round(altitude, 2)
16 print(altitude)
```

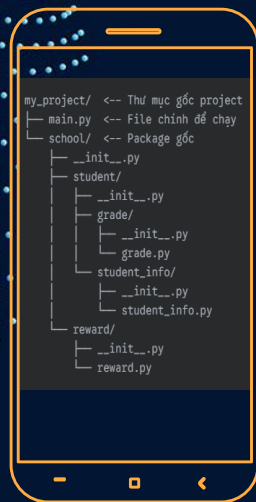
Cũng như con người các hàm có thể team-work cùng nhau, phối hợp lẫn nhau để hoàn thành nhiệm vụ.

SỬ DỤNG THƯ VIỆN

Về bản chất, thư viện là một **file chứa các hàm** đã lập trình sẵn, chỉ cần tải về rồi **import** vào code và sử dụng được ngay

```
1 # Gọi trực tiếp thư viện
2 # import math
3 # a = math.sqrt(9)
4 # p = math.pi
5
6 # Gọi trực tiếp thư viện và đổi tên
7 # import math as m
8 # a = m.sqrt(9)
9 # p = m.pi
10
11 # Gọi riêng chức năng cần dùng
12 from math import sqrt, pi
13 a = sqrt(9)
14 p = pi
15
16 # Gọi riêng chức năng cần dùng và đổi tên
17 from math import sqrt as cb2, pi as bi
18 # a = cb2(9)
19 # p = bi
20
21 print(a)
22 print(p)
```

SỬ DỤNG HÀM TỪ FILE KHÁC



Cùng Module

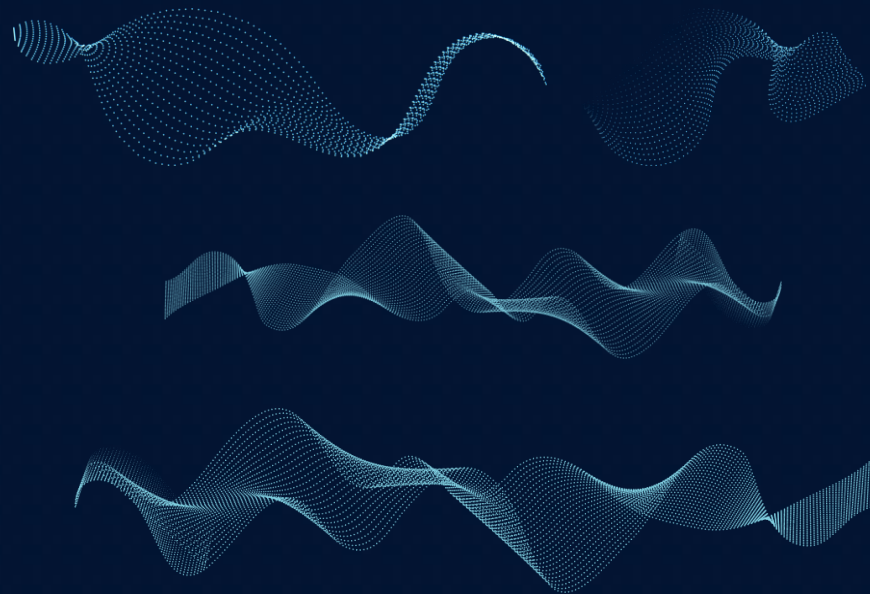
```
grade.py X
school > student > grade > grade.py > ...
1 # Import dữ liệu từ module student_info.py (đặt trong thư mục ngang cấp)
2 # Dùng import tương đối (relative import)
3 from ..student_info.student_info import students_data
```

Khác Module

```
reward.py X
school > reward > reward.py > ...
1 # Import hàm từ module grade.py (đặt ở một nhánh khác trong package school)
2 # Dùng import tuyệt đối (absolute import) từ package gốc 'school'
3 from school.student.grade.grade import avg_grade, grade_student_table
```

Bạn có thể coi file kia là một **thư viện** và **import** nó với đường dẫn chính xác vào code là sử dụng được ngay

LƯU Ý QUAN TRỌNG



Luôn luôn phải ghi nhớ:

- Hàm chỉ hoạt động khi có trong file và được gọi đến
- Hàm kết thúc khi gặp lệnh return hoặc chạy hết lệnh
- Đảm bảo giá trị đưa vào đầu vào đúng với định nghĩa của hàm và mục đích sử dụng



NỖ LỰC NHỎ TỪNG NGÀY TẠO NÊN THÀNH CÔNG LỚN

Bạn không cần cố gắng quá nhiều
mà chỉ cần luyện tập từng chút
một hàng ngày.
Đó là đủ đơn giản để thành công



04

ĐỆ QUY

Định nghĩa, ứng dụng,
khử đệ quy

**“Muốn hiểu đệ quy là gì,
phải hiểu đệ quy
là gì đã...?”**

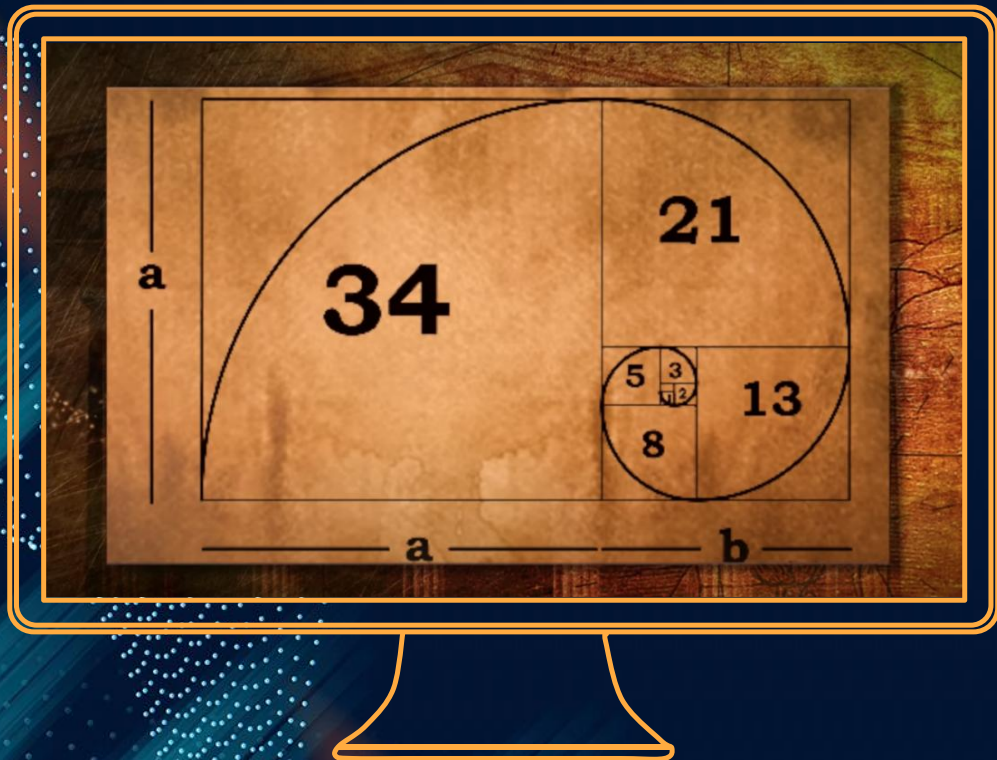
—Sưu tầm



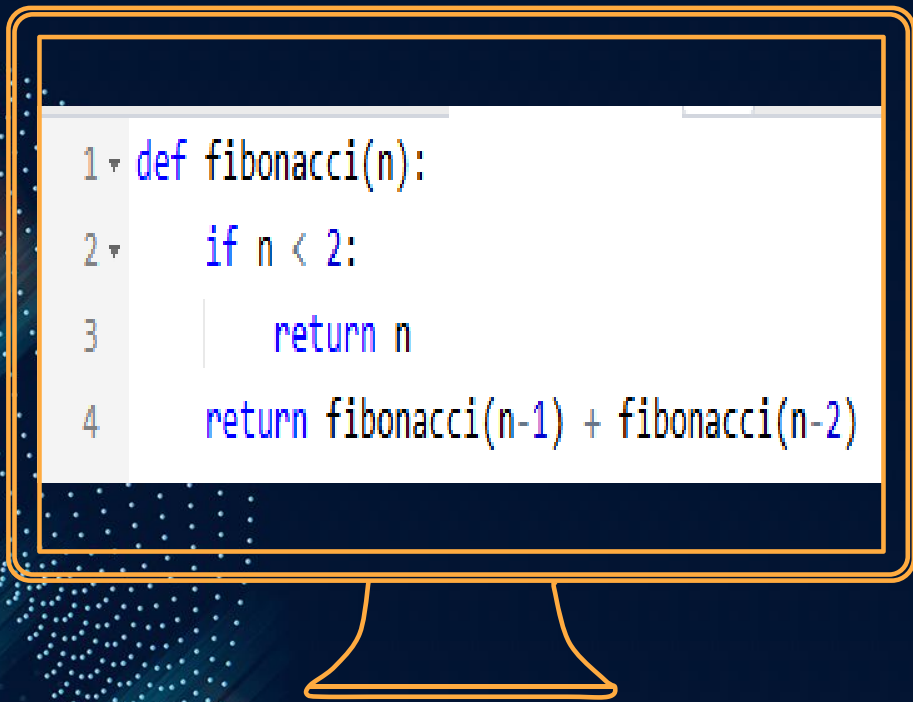
SỰ LẶP LẠI CẤU TRÚC TỪ LỚN ĐẾN NHỎ DẦN

Muốn tạo được cái lớn
phải có cái nhỏ hơn,
muốn có cái nhỏ hơn
phải có cái nhỏ hơn nữa

Dãy Fibonacci



Giải thuật đệ quy Fibonacci

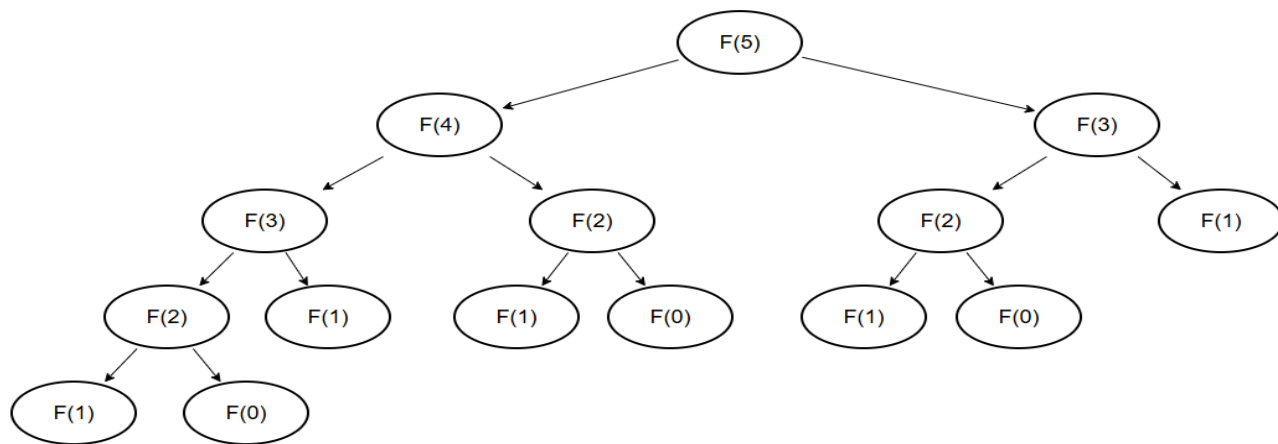


```
1 def fibonacci(n):  
2     if n < 2:  
3         return n  
4     return fibonacci(n-1) + fibonacci(n-2)
```

NGUYÊN LÝ:

- Dễ thấy các số từ thứ 3 trở đi đều có chung cách xác định. Do đó, ta chỉ cần 1 hàm **tự gọi chính mình** để tìm 2 giá trị trước nó, cho đến khi gọi đến 2 phần tử đầu tiên là 0 và 1 thì có thể tính ngược về giá trị cần tìm

Sơ đồ hoạt động của đệ quy thuần



ƯU – NHƯỢC ĐIỂM

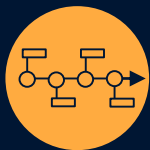
ƯU ĐIỂM

- Trực quan, rõ ràng, ngắn gọn, dễ hiểu, dễ triển khai
- Phù hợp với các cấu trúc dữ liệu phân cấp tương đồng

NHƯỢC ĐIỂM

- Tiêu tốn bộ nhớ khá nhiều do cần nhiều stack đệ quy
- Hiệu suất kém do tiêu tốn bộ nhớ và khả năng bị tính toán trùng lặp, đòi hỏi khả năng tối ưu tốt

CẢI THIẾN ĐỆ QUY



ĐỆ QUY ĐUÔI

Dùng tham số lưu trữ để truyền kết quả của các bước trước vào lời gọi đệ quy tiếp theo



LƯU NHỚ

Lưu trữ các kết quả của lời gọi đệ quy để tránh tính lại



QUY HOẠCH ĐỘNG

Tính từ dưới lên cho đến bước cần xác định kết quả

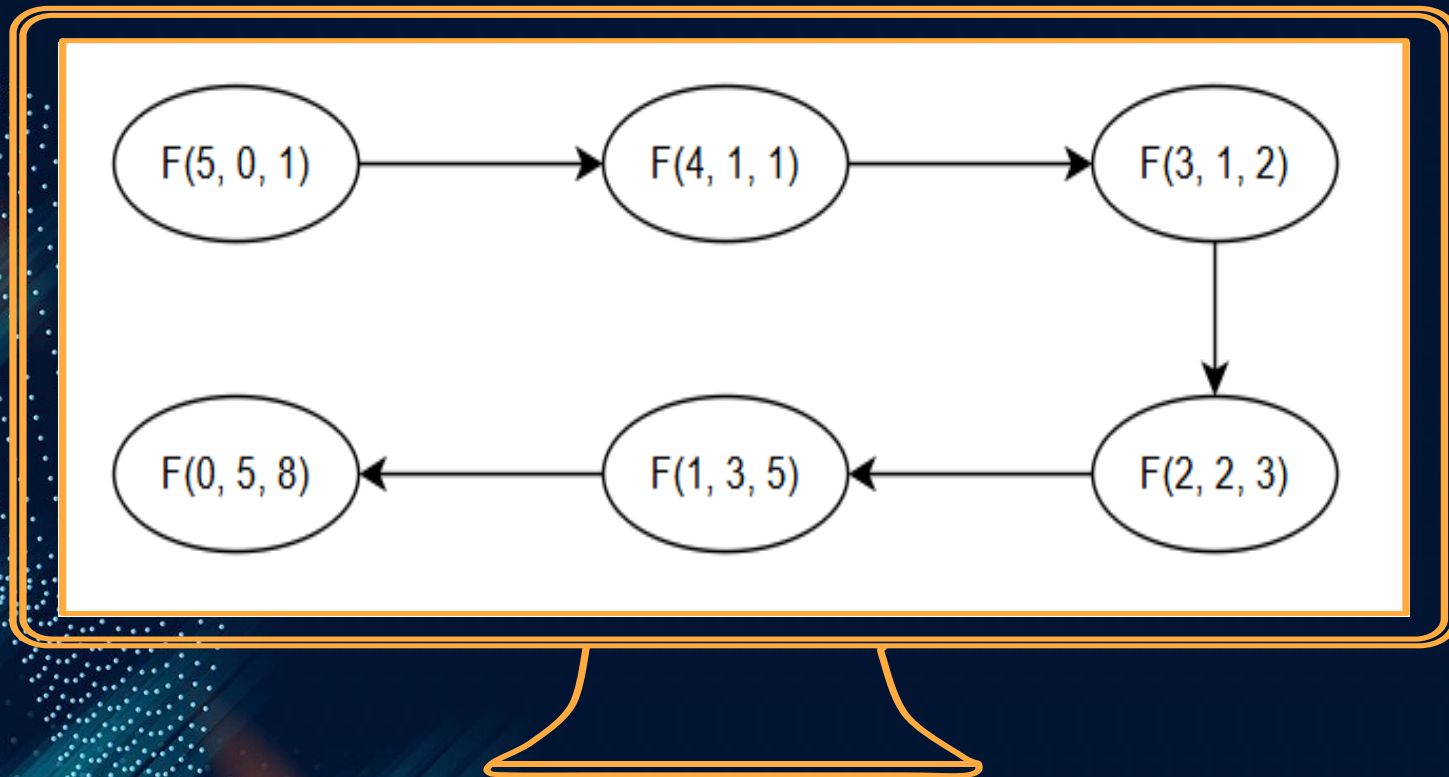
Đệ quy đuôi

```
1 def fib_tail(n, a=0, b=1):  
2     if n == 0:  
3         return a  
4     return fib_tail(n - 1, b, a + b)
```

NGUYÊN LÝ:

- Dùng 2 biến tích lũy (a, b) để lưu lại kết quả của bước trước rồi truyền cho bước sau
- Khi n chạy về 0, biến a sẽ chứa kết quả cần tìm

Sơ đồ hoạt động của đệ quy đuôi



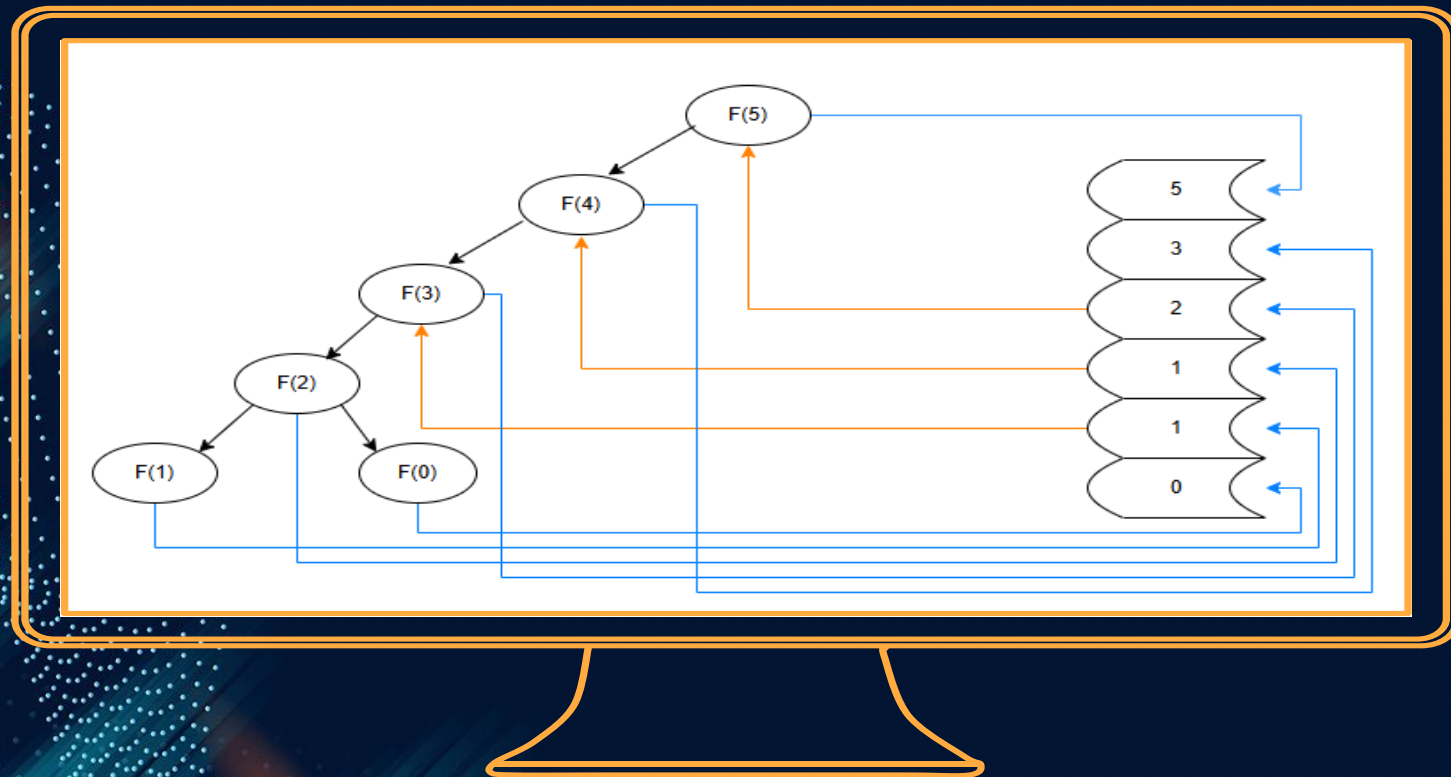
Lưu nhớ

```
1 def fib_memo(n, memo={}):  
2     if n in memo:  
3         return memo[n]  
4     if n <= 1:  
5         return n  
6     memo[n] = fib_memo(n - 1, memo) + fib_memo(n - 2, memo)  
7     return memo[n]
```

NGUYÊN LÝ:

- Dùng 1 danh sách để ghi nhớ kết quả của các bước đệ quy trước đó
- Kết quả không cần phải bị tính lại
- Chiều tiếp cận từ cấp cao xuống thấp

Sơ đồ hoạt động của lưu nhớ



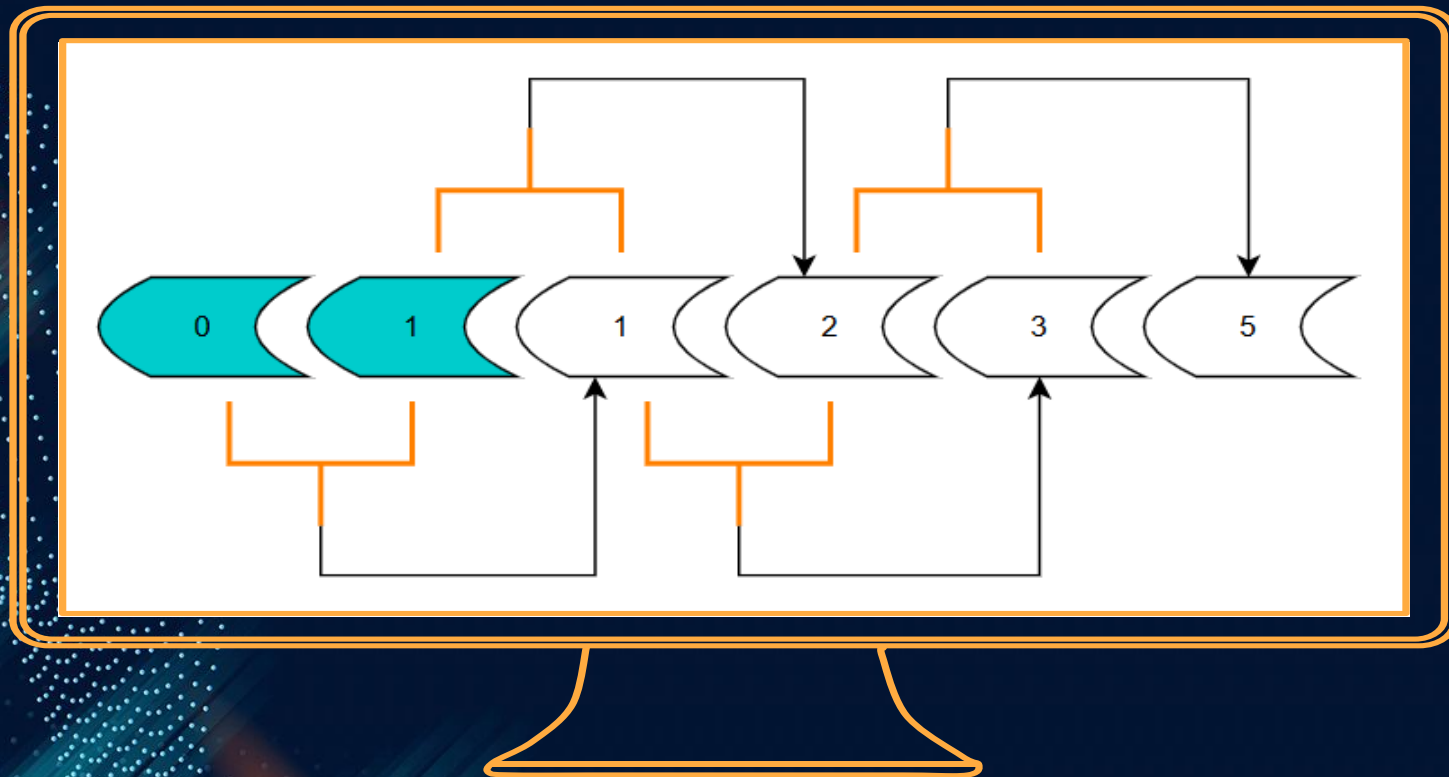
Quy hoạch động

```
1 ▾ def fib_dp(n):  
2 ▾     if n <= 1:  
3 ▾         return n  
4     dp = [0] * (n + 1)  
5     dp[1] = 1  
6 ▾     for i in range(2, n + 1):  
7 ▾         dp[i] = dp[i - 1] + dp[i - 2]  
8     return dp[n]
```

NGUYÊN LÝ:

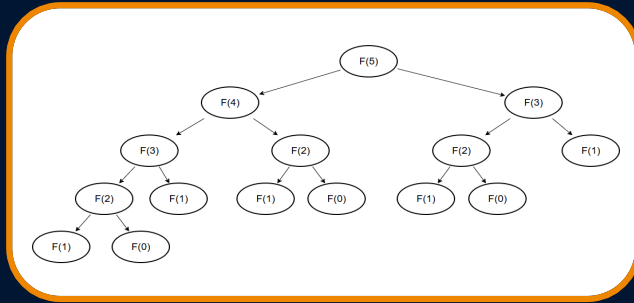
- Dùng danh sách lưu giá trị từ nhỏ đến lớn
- Không cần dùng tới tới đệ quy, dùng vòng lặp để thay cho lặp đệ quy
- Chiều tiếp cận từ cấp thấp lên cao

Sơ đồ hoạt động của quy hoạch động

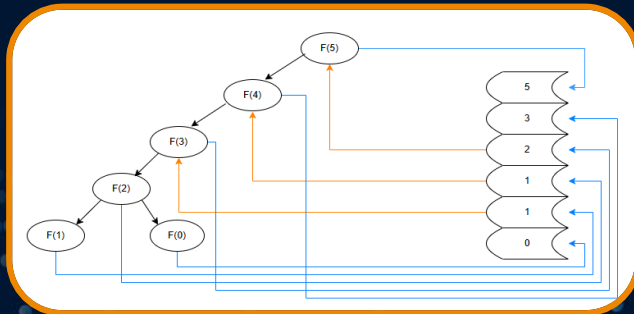
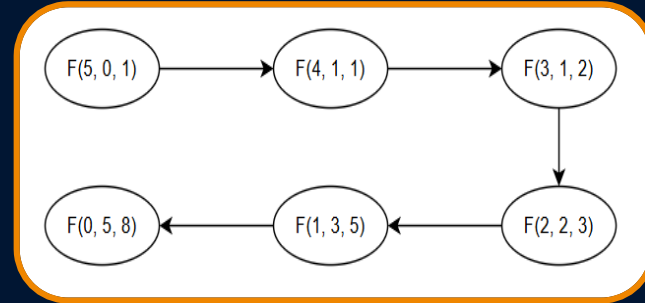


So sánh trên Fibonacci(5)

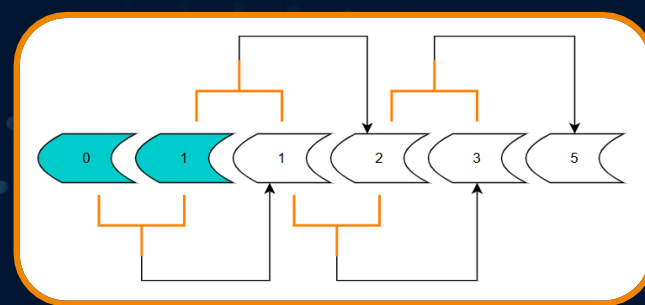
ĐỆ QUY THUẦN



ĐỆ QUY ĐUÔI



LƯU NHỚ



QUY HOẠCH ĐỘNG



SUCCESS

Thành công cũng là đệ quy của việc chinh
phục thách thức mới dựa trên nền tảng
chinh phục thách thức trước đó



05

KIỂM THỬ

Thử nghiệm, đọc code, dò lỗi,
gỡ lỗi, kiểm soát lỗi



THỬ NGHIỆM, ĐÁNH GIÁ VÀ CHỈNH SỬA

Những công đoạn cần
thiết và không thể thiếu
trong tiến trình hoàn
thiện sản phẩm

THỬ NGHIỆM

```
1 # Hàm tính cạnh huyền bằng Pythagoras
2 def pythagoras(edge1, edge2):
3     hypotenuse = (edge1**2 + edge2**2)**0.2
4     return hypotenuse
5
6 # Chọn dữ liệu đầu vào là dạng số học: 3 và 4
7 # Kết quả dự kiến: 5.0
8
9 # Kích hoạt hàm pythagoras() với đầy đủ 2 tham số đầu vào
10 hyp = pythagoras(3, 4)
11
12 # In kết quả ra Terminal để kiểm tra kết quả thực tế
13 print(hyp)
14
15 # So sánh kết quả dự kiến khác kết quả thực tế
16 # 1.9036539387158786 != 5.0
17
18 # Kết luận: Hàm hoạt động không chính xác yêu cầu
```

- Chọn đầu vào đúng yêu cầu của hàm
- Xác định kết quả dự kiến
- Kích hoạt hàm với đầu vào đã chuẩn bị
- In kết quả thực tế
- So sánh kết quả thực tế với kết quả dự kiến
- Kết luận

LUỒNG DỮ LIỆU

THỨ TỰ CHÍNH

Theo chiều từ trên xuống dưới

BỎ QUA

Các ghi chú và các khối hàm

LƯU Ý

Các lệnh gán khởi tạo không cần xét

HÀM

Chỉ chạy khi được gọi đến, dừng lại khi gặp lệnh return hoặc đã chạy hết lệnh

VÒNG LẶP

Chạy tuần tự cho đến khi hết số lần lặp hoặc gặp phải lệnh ngắt

LỆNH ĐIỀU KIỆN

Chỉ chạy lệnh của phần điều kiện đúng

LUỒNG DỮ LIỆU

```
1 students = [  
2     {'name': 'An', 'math': 8.2, 'physics': 9.5, 'chemistry': 8.5},  
3     {'name': 'Bình', 'math': 6.5, 'physics': 7.5, 'chemistry': 7},  
4     {'name': 'Cường', 'math': 9, 'physics': 9, 'chemistry': 10},  
5     {'name': 'Dương', 'math': 8, 'physics': 7, 'chemistry': 8.5},  
6 ]  
7  
8 # Kiểm tra điểm liệt  
9 def isValid(student):  
10     if student['math'] < 7 or student['physics'] < 7 or student['chemistry'] < 7: # 10  
11         return False  
12     else:  
13         return True # 11  
14  
15 # Tính điểm xét tuyển và xác nhận trúng tuyển  
16 def checkPassed(student):  
17     avg = round((student['math']*2 + student['physics'] + student['chemistry'])/4, 1) # 13  
18     if avg >= 8.1: # 14  
19         return (avg, 'trúng tuyển') # 15  
20     return (avg, 'không trúng tuyển')  
21  
22 def result(name, students):  
23     res = [] # 2  
24     for student in students: # 3 5 7  
25         if name == student['name']: # 4 6 8  
26             if isValid(student): # 9  
27                 res = checkPassed(student) # 12  
28             else:  
29                 return 'Bạn không trúng tuyển do điểm liệt'  
30             break # 16  
31     if res == []: # 17  
32         return 'Không có tên thí sinh này.'  
33     return f'Bạn đã {res[1]} với điểm xét tuyển là {res[0]}.' # 18  
34  
35 print(result('cường', students)) # 1
```

- Thứ tự chính
- Bỏ qua
- Lưu ý
- Hàm
- Vòng lặp
- Lệnh điều kiện

GỠ LỖI

THỬ NGHIỆM

- Trường hợp cơ bản:
Thử nghiệm xem code có hoạt động có hoạt động đúng yêu cầu cơ bản không
- Trường hợp đặc biệt:
Thử nghiệm các trường hợp đặc biệt, phát hiện các lỗi tiềm ẩn có thể xảy ra

LUỒNG DỮ LIỆU

- Đọc chính xác thứ tự vận hành của của các dòng code
- Kiểm tra cẩn thận các bước biến đổi dữ liệu, lệnh điều kiện, lệnh ngắt vòng lặp và lệnh gọi hàm vì đây là những vị trí dễ xảy ra lỗi nhất

Lỗi đầu vào

```
1 from math import sqrt
2 def pythagoras(edge1, edge2):
3     hyp = sqrt(edge1**2 + edge2**2)
4     return hyp
5
6 # Sai kiểu dữ liệu đầu vào
7 err1 = pythagoras('a', 'b')
8
9 # Thiếu tham số
10 err2 = pythagoras(3)
11
12 # Dữ liệu đầu vào không tồn tại
13 a = [6, 8]
14 err3 = pythagoras(a[1], a[2])
```

- Sai kiểu dữ liệu của tham số truyền vào
- Thừa hoặc thiếu tham số theo cấu tạo đầu vào của hàm
- Sử dụng dữ liệu đầu vào không tồn tại

Lỗi chức năng

```
1 isValid = False
2 while not isValid:
3     name = input('Nhập tên của bạn: ')
4     words = name.split()
5     if len(words) < 2:
6         continue
7     for i in range(len(words)):
8         # Đặt điều kiện không chuẩn
9         if words[i].isalpha():
10            break
11        if i == len(words) - 1:
12            isValid = True
13        # Đặt lệnh ngắt không hợp lý
14        break
15
16 def abbreviatedName(word):
17     abb = ''
18     for i in range(len(words) - 1):
19         # Dùng sai phép biến đổi dữ liệu
20         abb += words[0][i].upper() + '. '
21     # Lệnh đặt sai vị trí
22     abb += words[-1]
23     return abb
24
25 print(abbreviatedName(name))
```

- Dùng sai phép biến đổi dữ liệu
- Lệnh đặt sai vị trí
- Đặt điều kiện không chuẩn
- Dùng lệnh ngắt không hợp lý

KIỂM SOÁT LỖI

LỖI ĐẦU VÀO

- Thiết kế bộ dữ liệu nhất quán từ đầu về các quy ước đặt tên và truy xuất
- Kiểm soát dữ liệu đầu vào thông qua các lệnh **điều kiện**, đòi hỏi sự chính xác trước khi đưa dữ liệu vào hàm

LỖI CHỨC NĂNG

- Nên vẽ sơ đồ chức năng trước nhằm đánh giá hiệu quả của đoạn code, chỉnh sửa cải tiến hợp lý rồi mới triển khai
- Sử dụng các công cụ debug hay khối lệnh **xử lý ngoại lệ** để xử lý lỗi

Kiểm soát lỗi đầu vào

Muốn hàm không bị
“ngộ độc” thì tốt hơn
cả là ngay từ đầu hãy
đảm bảo nó được
“ăn uống sạch sẽ”

```
1  ''' Kiểm soát giá trị nhập vào từ bàn phím
2  chỉ gồm chữ và dấu cách, không có số và
3  các ký tự đặc biệt, tối thiểu phải có 2 từ
4  '''
5  isValid = False
6  while not isValid:
7      name = input('Nhập tên của bạn: ')
8      words = name.split()
9      if len(words) < 2:
10         continue
11     for i in range(len(words)):
12         if not words[i].isalpha():
13             break
14         if i == len(words) - 1:
15             isValid = True
16
17 def abbreviatedName(words):
18     abb = ''
19     for i in range(len(words) - 1):
20         abb += words[i][0].upper() + ' '
21     abb += words[-1]
22     return abb
23
24 print(abbreviatedName(words))
25
```

Kiểm soát lỗi chức năng

```
1 ''' Đọc, ghi file text có thể xảy ra nhiều trường hợp mắc lỗi khác nhau.  
2 Có những lỗi có thể lường trước được như không tìm thấy file, không có  
3 quyền đọc file, nhưng có những lỗi hiếm gặp mà có thể chúng ta không lường  
4 trước được khiến việc kiểm soát lỗi ban đầu trở nên khó khăn  
5 Vậy nên, bắt lỗi trong khi vận hành sẽ là khả thi hơn. '''  
6  
7 def readingText(filename):  
8     try:  
9         with open(filename, "r") as file:  
10             content = file.read()  
11             print(content)  
12     except FileNotFoundError:  
13         print("Lỗi: Tập không tồn tại. Vui lòng kiểm tra lại tên tập.")  
14     except PermissionError:  
15         print("Lỗi: Không có quyền đọc tập.")  
16     except Exception as e:  
17         print(f"Có lỗi xảy ra: {e}")  
18  
19 readingText('data.txt')
```

Đôi khi lưới lọc không thể lọc triệt để “**chất độc**” lọt vào hàm, hãy thiết lập thêm “**hệ thống miễn dịch**” để ứng phó với những tình huống như thế này

Điều kiện và Xử lý ngoại lệ

ĐIỀU KIỆN

- Cơ chế: kiểm tra trước, nếu không có lỗi mới thực hiện
- Ứng dụng: chuyên xử lý các lỗi thường gặp, có thể nhận biết trước, có tỷ lệ xảy ra cao

XỬ LÝ NGOẠI LỆ

- Cơ chế: thực hiện trước, nếu có lỗi mới xử lý sau
- Ứng dụng: chuyên dùng để xử lý các lỗi hiếm gặp, khó có thể nhận biết trước, có tỷ lệ xảy ra thấp

**“Khi code bị lỗi,
code không sai,
người tạo ra nó mới là
người sai.”**

—M'



06

PHONG CÁCH

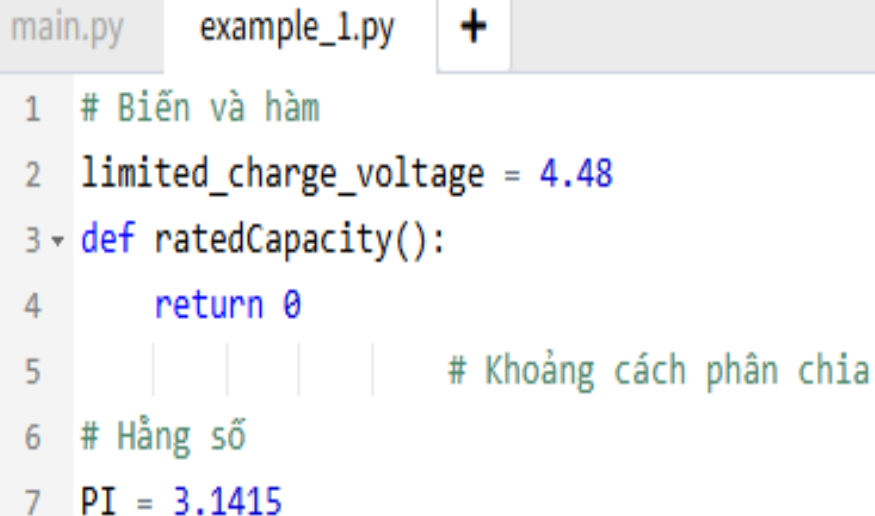
Quy ước đặt tên, bố trí dòng lệnh

```
web.1]: "protected":  
web.1]: "verified":  
web.1]: "followers_count":  
web.1]: "friends_count":  
web.1]: "listed_count":  
web.1]: "favourites_count":  
web.1]: "statuses_count":  
web.1]: "created_at":  
web.1]: "utc_offset":  
web.1]: "time_zone":  
web.1]: "geo_enabled":  
web.1]:
```

PHONG CÁCH LẬP TRÌNH KHOA HỌC

Những đoạn code được thiết kế khoa học sẽ dễ quan sát, dễ kiểm tra và dễ sửa chữa hơn

ĐẶT TÊN



```
main.py  example_1.py  +
1  # Biến và hàm
2  limited_charge_voltage = 4.48
3  def ratedCapacity():
4      return 0
5      |      |      |      |      # Khoảng cách phân chia
6  # Hằng số
7  PI = 3.1415
```

BIẾN, HÀM:

- snake_case
- camelCase

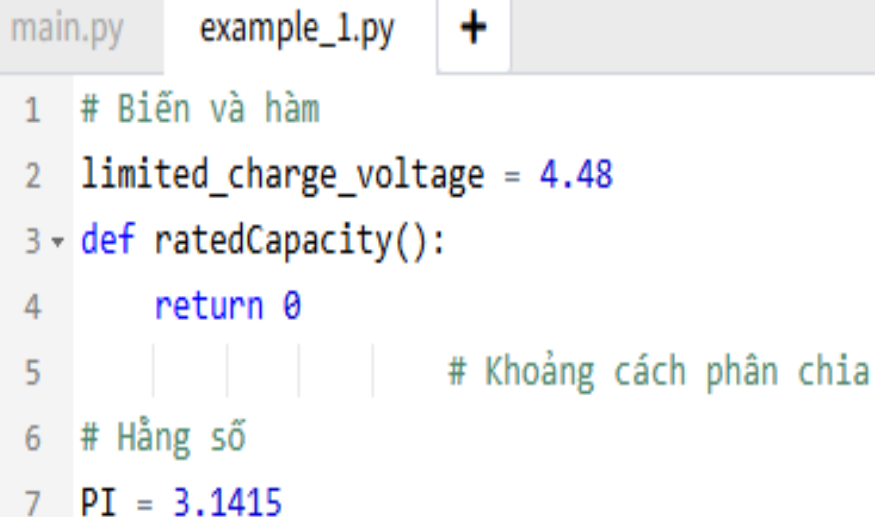
HẲNG SỐ:

- SNAKE_UPPER_CASE

FILE:

- snake_case.py
- PascalCase.js
- kebab-case.html

KHOẢNG CÁCH



```
main.py  example_1.py  +  
1  # Biến và hàm  
2  limited_charge_voltage = 4.48  
3  def ratedCapacity():  
4      return 0  
5      |   |   |   |   # Khoảng cách phân chia  
6  # Hằng số  
7  PI = 3.1415
```

GIẤN CÁCH KHỐI:

- Cách 1 dòng

ĐỘ DÀI DÒNG:

- Tối đa 80-100 ký tự

GIẤN CÁCH TOÁN TỬ:

- Cách 1 dấu cách

GHI CHÚ:

- Dòng ngay phía trên
- Cuối dòng

**“Công thức chung để
thành công trong mọi
lĩnh vực là đủ hiểu biết
cộng với đủ kiên trì.”**

—M’

CẢM ƠN!

Đăng ký ngay tại:
m.multiedu@gmail.com

Tiêu đề: IT - KTLT



Code your life!