



HCMUTE

TRƯỜNG ĐẠI HỌC

**SƯ PHẠM KỸ THUẬT TP. HỒ CHÍ MINH**

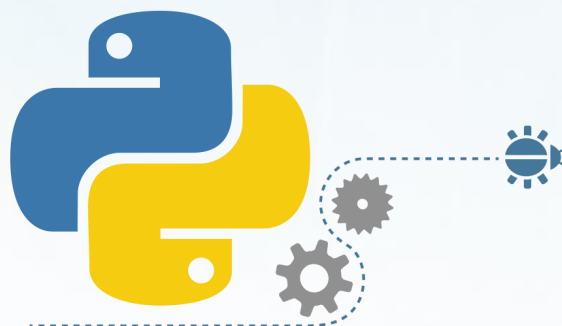
HCMC University of Technology and Education



KHOA CÔNG NGHỆ THÔNG TIN  
**BỘ MÔN HỆ THỐNG THÔNG TIN**

# **NHẬP MÔN LẬP TRÌNH PYTHON (IPPA233277)**

## **HÀM (FUNCTION)**



**GV. Trần Quang Khải**

1. Hiểu được khái niệm và nguyên tắc hoạt động về hàm
2. Biết cách viết hàm, gọi hàm
3. Sử dụng được Global variable, Parameter mặc định
4. Hiểu và thực hiện được hàm đệ quy
5. Sử dụng được một số hàm có sẵn của Python: các hàm toán học, round, time, random, exit, eval...
6. Nắm và vận dụng module



1. Khái niệm về hàm
2. Cấu trúc tổng quát của hàm
3. Cách gọi hàm
4. Nguyên tắc hoạt động
5. Phạm vi biến (variable scope)
6. Tham số mặc định (parameter)
7. Lambda expression
8. Hàm đệ quy
9. Một số hàm thông dụng
10. Module



- Hàm là một khối lệnh thực hiện một công việc hoàn chỉnh (module), được đặt tên và được gọi thực thi nhiều lần tại nhiều vị trí trong chương trình.
- Hàm được đặt tên, đầu vào, đầu ra, và có chức năng giải quyết một số vấn đề chuyên biệt cho chương trình chính, có thể được gọi nhiều lần (tái sử dụng) với các tham số khác nhau.
- Hàm còn gọi là chương trình con (subroutine)
- Có hai loại hàm:
  - ✓ Hàm thư viện: được xây dựng sẵn, muốn sử dụng cần được khai báo đầu chương trình
  - ✓ Hàm do người dùng tự định nghĩa
- Hàm phải luôn được định nghĩa trước khi gọi hàm, nếu không sẽ phát sinh lỗi



Ví dụ:

✓ Sử dụng hàm thư viện

```
print('Chương trình tính tổng hai số')
```

```
a, b = eval(input('Nhập a, b: '))
```

```
print("Tổng là: ", a + b)
```

✓ Hàm tự định nghĩa

```
def Tong(a, b):  
    return a + b
```



- Cấu trúc khai báo hàm:

```
def function_name(parameters):  
    """docstring"""  
    statement(s)  
    return expression
```

def *name* ( *parameter list* ) :  
 *block*

**Trong đó:**

- ✓ Từ khóa def để khai báo và định nghĩa một hàm
- ✓ function\_name là tên hàm (tuân thủ quy tắc đặt tên định danh)
- ✓ parameters là danh sách các tham số mà cung cấp giá trị đầu vào cho hàm (tùy chọn)
- ✓ Kết thúc khai báo hàm có dấu hai chấm :
- ✓ Thân hàm bắt đầu bằng thụt đầu dòng. Trong thân hàm có thể có nhiều câu lệnh (statement)
- ✓ Có thể sử dụng docstring để mô tả chức năng của hàm (tùy chọn)
- ✓ Một hàm có thể có (một hoặc nhiều) giá trị trả về với câu lệnh return hoặc dùng để thoát hàm (tùy chọn)





Từ khóa khai báo

Tên hàm Tham số đầu vào

```
def GiaiPTBac1(a, b): # khai báo hàm
```

```
    """Hàm giải phương trình bậc 1: ax + b = 0"""
```

docstring

```
    if a == 0 and b == 0:
```

```
        return 'Vô số nghiệm!'
```

```
    elif a == 0 and b != 0:
```

```
        return 'Vô nghiệm!'
```

```
    else: # a != 0
```

```
        return "x = {0}".format(round(-b/a, 2))
```

Hàm có trả về

Thân hàm

```
10 (function) def GiaiPTBac1(
11     a: Any,
12     b: Any
13 ) -> (str | None)
14     Hàm giải phương trình bậc 1: ax + b = 0
15     GiaiPTBac1
```



- Để gọi hàm cần kiểm tra hàm được định nghĩa như thế nào trước khi sử dụng
  - ✓ Có đối số đầu vào hay không?
  - ✓ Có trả về kết quả hay không?
- Nếu có kết quả trả về : `result = function_name([parameter])`
- Nếu không có kết quả trả về : `function_name([parameter])`

Ví dụ:

```
result = GiaiPTBac1(0, 0)
print(result) # Vô số nghiệm!
result = GiaiPTBac1(0, 8)
print(result) # Vô nghiệm!
result = GiaiPTBac1(5, 8)
print(result) # x = -1.6
```

```
def Hello(name):
    print("Hello", name)
Hello("Tèo") # Hello Tèo
```





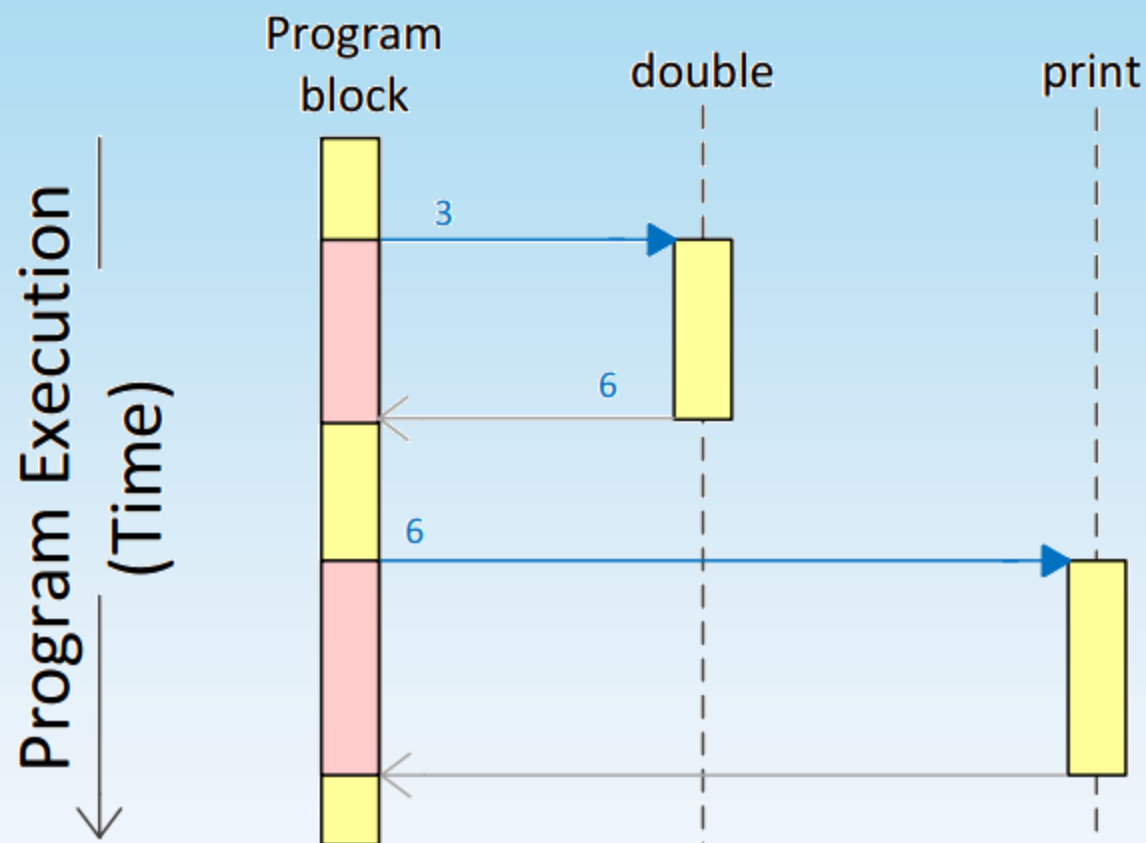
- Hàm trong python cũng như trong các ngôn ngữ lập trình khác đều hoạt động dựa theo nguyên tắc **LIFO** (Last – In – First – Out)

Ví dụ:

```
def double(n):
    return 2 * n

x = double(3)

print(x)
```



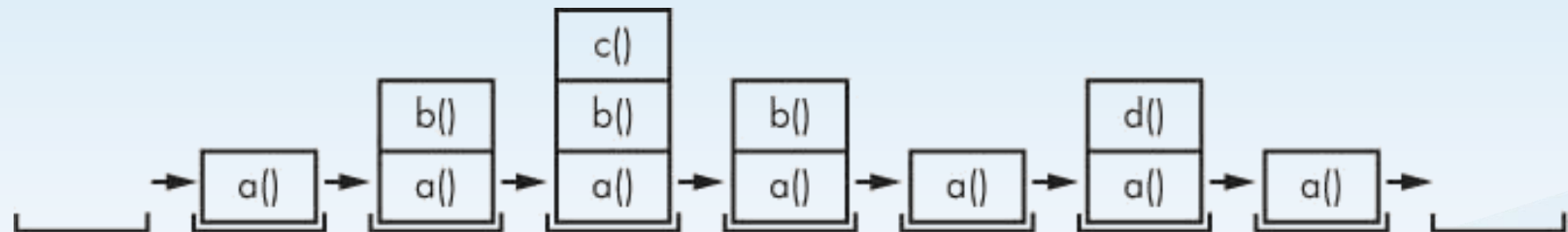
```
def a():
    print('a() starts')
    ❶ b()
    ❷ d()
    print('a() returns')
```

```
def b():
    print('b() starts')
    ❸ c()
    print('b() returns')
```

```
def c():
    ❹ print('c() starts')
    print('c() returns')
```

```
def d():
    print('d() starts')
    print('d() returns')
```

❺ a()



- Tất cả các biến khai báo trong hàm chỉ có phạm vi ảnh hưởng trong hàm, các biến này gọi là biến cục bộ (local variable). Khi thoát khỏi hàm thì các biến này không thể truy xuất được.
- Biến global là những biến được tạo ra không ở trong thân bất kỳ hàm nào mà ở phạm vi toàn cục (global scope).
- Biến toàn cục có phạm vi sử dụng trong toàn bộ chương trình, cả bên trong hoặc bên ngoài bất kỳ hàm nào

Ví dụ:

`n = 3` —————> **Biến toàn cục**

`def increase():`

`n = 2` —————> **Biến cục bộ**

`n = n + 1` —————> **Biến cục bộ**

`increase()`

`print(n)` `# ... ?` —————> **Biến toàn cục**



Ví dụ:

```
# global variable  
s = "python.org"  
def myFunction():  
    #local variable  
    s = s + "is great!"  
    print(s)  
myFunction()
```

➔ ...?

➔ *Biến trong hàm dù đã được khai báo bên ngoài thì vẫn được xem là biến cục bộ!*



- Nếu muốn chỉ định một biến trong hàm là biến toàn cục chứ không phải là biến cục bộ → sử dụng từ khóa **global**
- Từ khóa **global** cho phép tham chiếu sử dụng được biến toàn cục

```
# global variable
```

```
s = "python.org"
```

```
def myFunction():
```

```
    global s
```

```
    s = s + " is great!"
```

```
    print("Inside function:", s) # Inside function: python.org is great!
```

```
myFunction()
```

```
print("Outside function:", s) # Outside function: python.org is great!
```



- Biến **nonlocal** trong Python: **nonlocal** là một từ khóa trong Python được sử dụng để tham chiếu đến biến trong phạm vi cha của một hàm.
- Trong Python, khi một biến được khai báo trong một hàm, biến này sẽ có phạm vi là biến cục bộ của hàm đó. Tuy nhiên, khi một hàm được định nghĩa bên trong một hàm khác, biến cục bộ của hàm ngoài cũng không thể được truy cập từ bên trong hàm lồng nhau.
- Trong trường hợp này, chúng ta có thể sử dụng từ khóa **nonlocal** để tham chiếu đến biến trong phạm vi cha của hàm lồng nhau. Từ khóa `nonlocal` cho phép gán giá trị mới cho biến trong phạm vi cha của hàm lồng nhau.





Ví dụ:

```
def outer():  
    x = "local"  
    def inner():  
        nonlocal x  
        x = "nonlocal"  
        print("inner x:", x) # inner x: nonlocal  
    inner()  
    print("outer x:", x) # outer x: nonlocal  
outer()
```

➔ Nếu thay đổi giá trị của một biến **nonlocal** thì biến cục bộ cũng sẽ thay đổi



- Python cũng tương tự như C++, có hỗ trợ tham số mặc định khi khai báo hàm
- Tham số được gán giá trị trong khai báo hàm được gọi là giá trị mặc định cho tham số

Ví dụ:

```
def faculty(name = "Information Technology"):
```

```
    print("Welcome to", name)
```

```
faculty()                # Welcome to Information Technology
```

```
faculty("Economics")    # Welcome to Economics
```

- Python sử dụng / và \* để cụ thể một hàm chỉ có thể truyền giá trị theo vị trí (cho /, đứng sau tất cả các tham số) hay keyword (cho \*, đứng trước tất cả các tham số)

```
def my_function(x, /):
```

```
    print(x)
```

```
my_function(3)
```

```
def my_function(*, x):
```

```
    print(x)
```

```
my_function(x = 3)
```



- Nếu không biết có bao nhiêu đối số được truyền vào hàm, thêm \* vào trước tham số trong định nghĩa hàm. Hàm sẽ hiểu tham số này có dạng tuple để truy xuất
- Nếu không biết có bao nhiêu tham số được truyền vào hàm, thêm \*\* vào trước tham số trong định nghĩa hàm. Hàm sẽ hiểu tham số này có dạng dictionary để truy xuất

Ví dụ:

**# Arbitrary Arguments, \*args**

```
def my_function(*kids):  
    print("The youngest child is " + kids[2])  
my_function("Emil", "Tobias", "Linus")
```

**# Arbitrary Keyword Arguments, \*\*kwargs**

```
def my_function(**kid):  
    print("His last name is " + kid["lname"])  
my_function(fname = "Tobias", lname = "Refsnes")
```



- Python hỗ trợ kiểu khai báo hàm nặc danh thông qua biểu thức lambda

**lambda** *parameterlist* : *expression*

- Hàm lambda là hàm không có tên, còn được gọi là hàm ẩn danh (anonymous function).
- Hàm lambda có số lượng tham số bất kỳ nhưng chỉ có một biểu thức. Biểu thức sẽ tính toán, đánh giá, trả kết quả
- Hàm lambda có thể được sử dụng ở bất cứ nơi nào được yêu cầu

## Trong đó:

- ✓ lambda : từ khóa
- ✓ parameterlist : tập hợp các parameter mà ta muốn định nghĩa
- ✓ expression : biểu thức đơn trong python (không nhập complex)



✓ Định nghĩa một hàm

```
def handle(f, x):
    return f(x)
```

✓ Đối số của hàm trên là một hàm f nào đó với giá trị x đi kèm

✓ Từ hàm handle này, có thể gọi tùy ý các giao tác:

```
ret1 = handle(lambda x : x % 2 == 0, 7)
```

```
ret2 = handle(lambda x : x % 2 != 0, 7)
```

✓ Trước dấu hai chấm (:) là từ khóa lambda, phía sau là số lượng các biến được khai báo trong handle (tính sau chữ f). Tức nếu là handle(f, x, y) thì viết lambda x, y:

```
def handle(f, x, y):
    return f(x, y)
```

```
sum = handle(lambda x, y: x + y, 7, 9)
```

```
print(sum) # ...?
```



- Vì lambda expression không nhận cách viết complex, do đó muốn complex thì cần định nghĩa một hàm độc lập rồi truyền cho lambda

**Ví dụ:** Cho các hàm sau

```
def handle(f, x):
    return f(x)

def SoChan(x):
    return x % 2 == 0

def SoLe(x):
    return x % 2 == 1

def SoNguyenTo(x):
    dem = 0
    for i in range(1, x + 1):
        if x % i is 0:
            dem += 1
    return dem == 2
```

```
ret1 = handle(SoChan, 6)
print(ret1) # ...?

ret2 = handle(lambda x : SoChan(x), 6)
print(ret2) # ...?

ret3 = handle(SoLe, 6)
print(ret3) # ...?

ret4 = handle(lambda x : SoLe(x), 7)
print(ret4) # ...?

ret5 = handle(SoNguyenTo, 5)
print(ret5) # ...?

ret6 = handle(lambda x : SoNguyenTo(x), 9)
print(ret6) # ...?
```





- Đệ quy là cách mà hàm tự gọi lại chính nó, trong nhiều trường hợp đệ quy giúp giải quyết những bài toán lớn. Tuy nhiên đệ quy nếu xử lý không khéo sẽ bị tràn bộ đệm.
- Thông thường ta nên cố gắng giải quyết bài toán đệ qui bằng các vòng lặp, khi nào không thể giải quyết bằng vòng lặp thì mới nghĩ tới đệ quy. Do đó có những bài toán **Khử đệ quy** thì nên nghĩ về các vòng lặp để khử.
- Một vài ví dụ kinh điển về đệ quy như tính giai thừa, tính dãy số Fibonacci.

✓  $N! = N * (N - 1)!$  → Đệ qui: Nếu biết được  $(N - 1)!$  thì sẽ tính được  $N!$

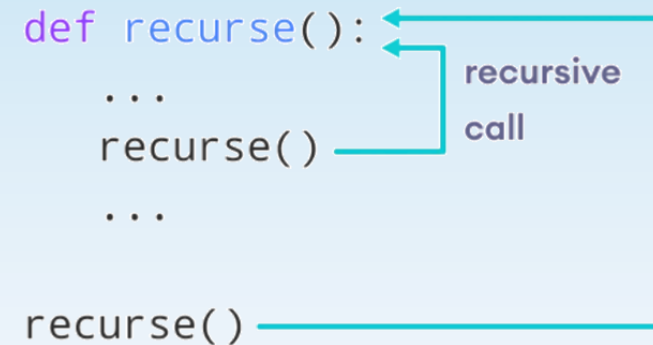
✓  $F_1 = 1, F_2 = 1, F_N = F_{N-1} + F_{N-2}$

- Cấu trúc hàm đệ quy (hình bên)

Lưu ý:

- Không thể để hàm gọi hàm liên tục (phải có điều kiện dừng)
- Trình thông dịch python giới hạn độ sâu của đệ quy (depths of recursion) để tránh đệ quy vô hạn dẫn đến tràn ngăn xếp (stack), mặc định 1000, nếu vi phạm sẽ phát sinh lỗi **RecursionError**

```
def recurse():
    ...
    recurse()
    ...
recurse()
```




**Ví dụ:** Phân tích bài toán giai thừa theo cách đệ quy

$$n! = n * (n - 1) * (n - 2) * (n - 3) \dots 3 * 2 * 1$$

Viết lại dạng phương trình đệ quy có điều kiện:

$$n! = \begin{cases} 1, & \text{if } n = 0 \\ n * (n - 1)! \end{cases}$$

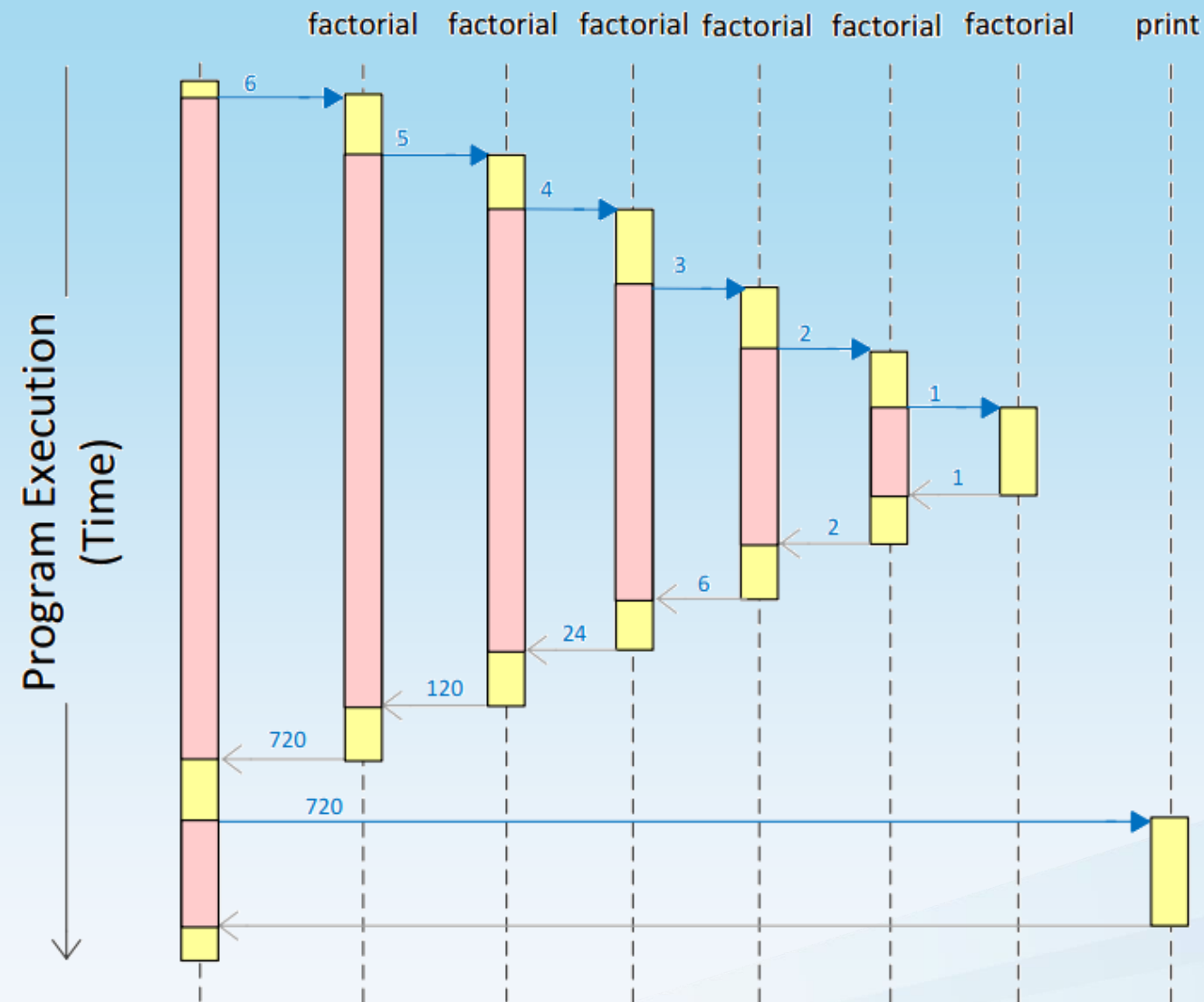
Điểm dừng là khi  $n = 0$ , quy luật nếu biết  $(n - 1)!$  thì tính được  $n!$  vì  $n! = n * (n - 1)!$

```
def factorial(n):  
    """Hàm tính n!"""  
    if n == 0:  
        return 1  
    else:  
        return n * factorial(n - 1)  
  
result = factorial(6)  
print("6! =", result)
```



```

6! = 6 * factorial(5)
    = 6 * 5 * factorial(4)
    = 6 * 5 * 4 * factorial(3)
    = 6 * 5 * 4 * 3 * factorial(2)
    = 6 * 5 * 4 * 3 * 2 * factorial(1)
    = 6 * 5 * 4 * 3 * 2 * 1 * factorial(0)
    = 6 * 5 * 4 * 3 * 2 * 1 * 1
    = 6 * 5 * 4 * 3 * 2 * 1
    = 6 * 5 * 4 * 3 * 2
    = 6 * 5 * 4 * 6
    = 6 * 5 * 24
    = 6 * 120
    = 720
    
```



- Các dạng hàm toán học
- Hàm Round
- Hàm DateTime
- Hàm Random
- Hàm Exit
- Hàm Eval



```
from math import *
```

- Tính căn bậc 2 – sqrt      `print('sqrt(25) =', sqrt(25))`      # sqrt(25) = 5.0
- Tính lũy thừa – pow      `print('pow(5, 2) =', pow(5, 2))`      # pow(5, 2) = 25.0
- Tính logarit cơ số e – log(x)      `print('log(2) = ', log(2))`      # log(2) = 0.693147180
- Tính logarit cơ số 10 – log10(x)      `print('log10(100) =', log10(100))`      # log10(100) = 2.0
- Tính  $e^x$  – exp(x)      `print('exp(2) = ', exp(2))`      # exp(2) = 7.38905609
- Đổi radian  $180/\pi x$  – radians      `print(radians(30))`      # 0.5235987755982988  
– degrees      `print(degrees(0.5235987755982988))`      # 29.999999999999996
- Tính giá trị tuyệt đối – fabs      `print('fabs(-2) =', fabs(-2))`      # fabs(-2) = 2.0
- Với các hàm lượng giác, cần chuyển đổi từ góc về radian sau đó mới sử dụng sin, cos, tan,...



- Hàm làm tròn các số lẻ phần thập phân

Cú pháp:

`round(<value>, <precision>)`

```
a = 3
```

```
b = 11
```

```
c = b/a
```

```
print(c)           # 3666.6666666666665
```

```
print(round(c, 2)) # 3666.67
```

```
print(round(c, 0)) # 3667.0
```

```
print(round(c, -2)) # 3700.0
```





- Ngày không phải là một kiểu dữ liệu của python nên cần sử dụng thư viện datetime

```
import datetime
```

```
x = datetime.datetime.now() # hoặc x = datetime.datetime(2020, 5, 17)
```

```
print(x)
```

- Định dạng hiển thị thời gian sử dụng phương thức strftime()

```
import datetime
```

```
x = datetime.datetime(2018, 6, 1)
```

```
print(x.strftime("%B"))
```



	MÔ TẢ	VÍ DỤ		MÔ TẢ	VÍ DỤ
%a	Weekday, short version	Wed	%y	Year, short version, without century	18
%A	Weekday, full version	Wednesday	%Y	Year, full version	2018
%w	Weekday as a number 0-6, 0 is Sunday	3	%H	Hour 00-23	17
%d	Day of month 01-31	31	%I	Hour 00-12	05
%b	Month name, short version	Dec	%p	AM/PM	PM
%B	Month name, full version	December	%M	Minute 00-59	41
%m	Month as a number 01-12	12	%S	Second 00-59	08



	MÔ TẢ	VÍ DỤ		MÔ TẢ	VÍ DỤ
%f	Microsecond 000000-999999	548513	%C	Century	20
%z	UTC offset	+0100	%x	Local version of date	12/31/18
%Z	Timezone	CST	%X	Local version of time	17:41:00
%j	Day number of year 001-366	365	%%	A % character	%
%U	Week number of year, Sunday as the first day of week, 00-53	52	%G	ISO 8601 year	2018
%W	Week number of year, Monday as the first day of week, 00-53	52	%u	ISO 8601 weekday (1-7)	1
%c	Local version of date and time	Mon Dec 31 17:41:00 2018	%V	ISO 8601 weeknumber (01-53)	01



- Random là một trong những hàm khá hữu dụng trong việc khởi tạo các giá trị ngẫu nhiên, phát sinh giả lập dữ liệu, thống kê

Cú pháp:

```
import random  
  
# lấy số ngẫu nhiên trong [start, stop) với step  
random.randrange(start, stop, step)
```



- Hàm `exit()` dùng để thoát phần mềm

```
while True:
```

```
    s = input("Tên bạn:")
```

```
    print(s)
```

```
    hoi = input("Tiếp tục? (c/k):")
```

```
    if hoi == "k":
```

```
        exit()
```

```
print("BYE!")
```



- Hàm **eval()** cho phép tự tính toán chuỗi phép toán / chuỗi lệnh

```
from math import sin
```

```
x = eval("1 + 2 + 5 + sin(30)")
```

```
print(x)
```

```
# hoặc
```

```
x1, x2 = eval(input("Nhập x1, x2: "))
```

```
print("x1 =", x1, ", x2 =", x2)
```

```
print("{0} + {1} = {2}".format(x1, x2, x1 + x2))
```





- Module trong python là một tập tin chứa những câu lệnh (statement) và định nghĩa (definition)
- Trong module, có thể định nghĩa hàm (function), lớp (class), biến (variable). Do đó, có thể chia chương trình lớn thành các tập tin, giúp dễ quản lý chương trình và tổ chức cấu trúc chương trình logic hơn, tái sử dụng code
- Để sử dụng các thành phần của module này cho module khác, cần khai báo ở đầu tập tin bằng import, sau đó sử dụng toán tử chấm (.) để sử dụng các thành phần của module được import

Ví dụ:

**example.py**

```
def add(x, y):  
    result = x + y  
    return result
```

**main.py**

```
import example  
  
a = 1  
b = 2  
  
print("a + b = ", example.add(a, b))
```



- Import và đổi tên module khi quá dài, sử dụng từ khóa `as`

```
import math as m  
  
print("The value of pi is", m.pi)
```

- Có thể import thành phần cụ thể từ module mà không cần import toàn bộ

```
from math import pi, e  
  
print("The value of pi is", pi)
```

- Import tất cả các thành phần của module với dấu hoa thị (\*)

```
from math import *  
  
print("The value of pi is", pi)
```



1. Để hàm trả về giá trị ta sử dụng từ khóa?
  - A. return
  - B. exit
  - C. break
  - D. continue
2. Dấu “""" ... """” sau định nghĩa hàm là?
  - A. docstring
  - B. comment
  - C. note
  - D. Không ý nghĩa
3. Python có 2 loại hàm chính, đó là:
  - A. Custom function & User defined function
  - B. Built-in function & User defined function
  - C. Built-in function & User function
  - D. System function & User function
4. Hàm được khai báo ở đâu?
  - A. Module
  - B. Class
  - C. Trong một hàm khác
  - D. Tất cả đều đúng
5. Khi sử dụng hàm sqrt() ta cần khai báo thư viện:
  - A. math
  - B. random
  - C. zlib
  - D. datetime
6. Một hàm có thể được thực hiện với những giá trị do chương trình truyền vào qua
  - A. Điều kiện hàm
  - B. Lời gọi hàm
  - C. Dữ liệu đầu vào
  - D. File dữ liệu
7. Đây là lợi thế của việc sử dụng hàm trong Python?
  - A. Tránh việc phải lặp lại code thực thi những tác vụ tương tự nhau
  - B. Phân tách các vấn đề phức tạp thành các phần đơn giản hơn
  - C. Code rõ ràng, dễ quản lý hơn
  - D. Tất cả các đáp án đều đúng



- ✓ Họ tên : **Trần Quang Khải**
- ✓ Email : **khaitq@hcmute.edu.vn**
- ✓ Zalo (mã Qr)

