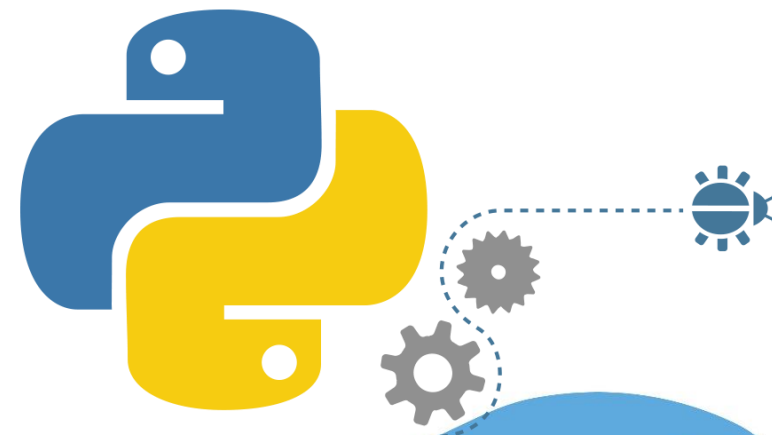


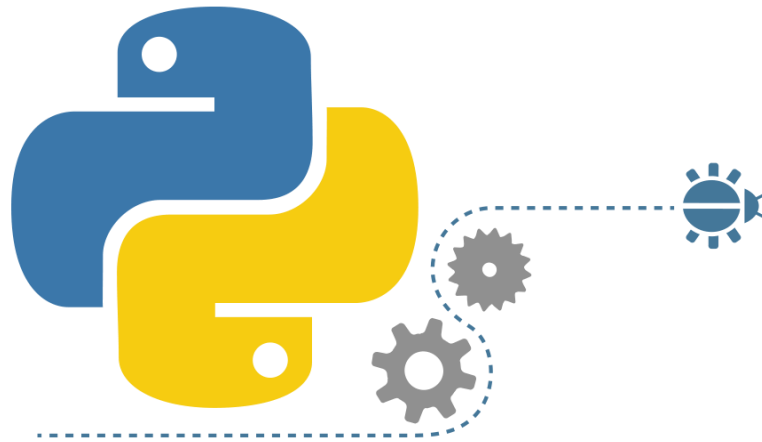


Môn Học **KỸ THUẬT LẬP TRÌNH** **VỚI PYTHON**

GV: Ths. Trần Duy Thanh
thanhtd@uel.edu.vn



HÀM TRONG PYTHON





Mục tiêu bài học

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



Nội dung bài học

- 4.1. Khái niệm về hàm
- 4.2. Cấu trúc tổng quát của hàm
- 4.3. Cách gọi hàm
- 4.4. Nguyên tắc hoạt động của hàm
- 4.5. Viết tài liệu cho hàm
- 4.6. Global Variable
- 4.7. Parameter mặc định
- 4.8. Lambda Expression
- 4.9. Giới thiệu về hàm đệ qui
- 4.10. một số hàm quan trọng thường dùng
 - 4.10.1. Các hàm toán học
 - 4.10.2. round
 - 4.10.3. Time
 - 4.10.4. Random
 - 4.10.5. exit
 - 4.10.6. eval



Nội dung bài học

4.1. Khái niệm về hàm

4.2. Cấu trúc tổng quát của hàm

4.3. Cách gọi hàm

4.4. Nguyên tắc hoạt động của hàm

4.5. Viết tài liệu cho hàm

4.6. Global Variable

4.7. Parameter mặc định

4.8. Lambda Expression

4.9. Giới thiệu về hàm đệ qui

4.1. Khái niệm về hàm

- 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 còn gọi là chương trình con (*subroutine*)
- ***Nếu không viết hàm thì sẽ gặp những khó khăn gì?***
 - Rất khó để viết chính xác khi dự án lớn
 - Rất khó debug
 - Rất khó mở rộng

4.1. Khái niệm về hàm

- Có hai loại hàm:
 - *Hàm thư viện*: là những hàm đã được xây dựng sẵn. Muốn sử dụng các hàm thư viện phải khai báo thư viện chứa nó trong phần khai báo `from ... import`.
 - *Hàm do người dùng định nghĩa*.

4.1. Khái niệm về hàm

- Ví dụ hàm thư viện:

```
print("Chương trình tính điểm trung bình")
toan, ly, hoa = eval(input("Nhập điểm toán, lý, hóa: "))
print("Điểm toán=", toan)
print("Điểm lý=", ly)
print("Điểm hóa=", hoa)
dtb = (toan + ly + hoa) / 3
print("Điểm trung bình=", dtb)
print("Điểm làm tròn=", round(dtb, 2))
```

- Ví dụ hàm tự định nghĩa:

```
def cong(x, y):
    return x + y
```


4.2. Cấu trúc tổng quát của hàm

- Python có cấu trúc tổng quát khi khai báo Hàm như sau:

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

- Dùng từ khóa **def** để định nghĩa hàm, các hàm có thể có đối số hoặc không. Có thể trả về kết quả hoặc không

4.2. Cấu trúc tổng quát của hàm

- Ví dụ : Viết hàm tính giải phương trình bậc 1

```
1  def PTB1(a,b):  
2      if a ==0 and b==0:  
3          return "Vô số nghiệm"  
4      elif a==0 and b!=0:  
5          return "Vô nghiệm"  
6      else:  
7          return "x={0}".format(round(-b/a,2))
```

- Ví dụ: Viết hàm xuất dữ liệu ra màn hình

```
def XuatDuLieu(data):  
    print(data)
```



4.3. Cách gọi hàm

- Để gọi hàm ta cũng cần phải kiểm tra Hàm đó được định nghĩa như thế nào?
 - ✓ Có đối số hay không?
 - ✓ Có trả về kết quả hay không?

Nếu có kết quả trả về:

Result=FunctionName ([parameter])

Nếu không có kết quả trả về:

FunctionName([parameter])

4.3. Cách gọi hàm

- Ở bài học trước ta có ví dụ về phương trình bậc 1 và xuất dữ liệu

```
def PTB1(a,b):  
    if a ==0 and b==0:  
        return "Vô số nghiệm"  
    elif a==0 and b!=0:  
        return "Vô nghiệm"  
    else:  
        return "x={0}".format(round(-b/a,2))
```

- Hàm PTB1 vừa có đối số vừa có kết quả trả về. Ta có thể gọi như sau:
kq=PTB1(5,8)



4.3. Cách gọi hàm

➤ Hàm xuất dữ liệu

```
def XuatDuLieu(data):  
    print(data)
```

Ta có thể gọi:

XuatDuLieu("hello")

4.4. Nguyên tắc hoạt động của hàm

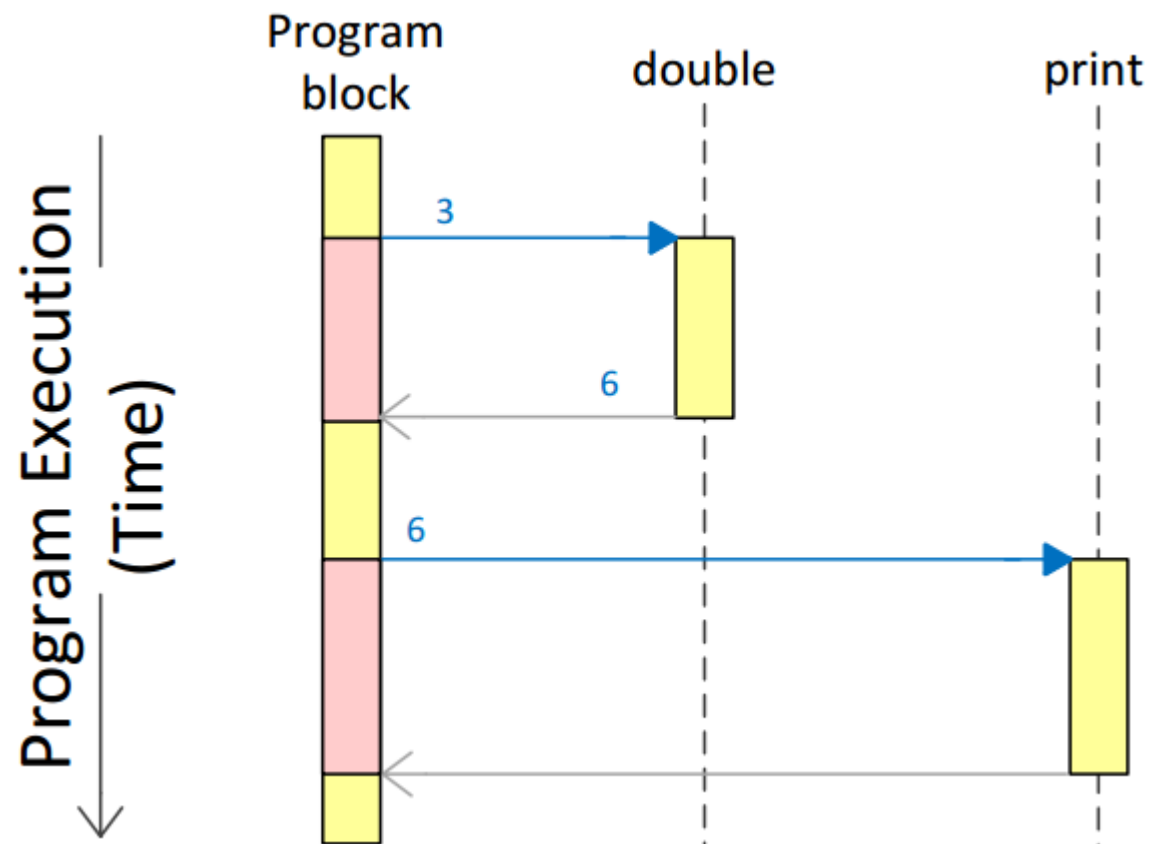
- Hàm trong Python cũng như trong các ngôn ngữ lập trình khác, đều hoạt động theo Nguyên tắc LIFO (LAST IN FIRST OUT)
- Ví dụ ta có các mã lệnh dưới đây:

```
def double(n):  
    return 2 * n  
  
x = double(3)  
print(x)
```

- Theo LIFO thì nó xảy ra như thế nào?

4.4. Nguyên tắc hoạt động của hàm

- Minh họa nguyên tắc hoạt động khi gọi hàm



```
def double(n):  
    return 2 * n  
x = double(3)  
print(x)
```

4.5. Viết tài liệu cho hàm

- Python hỗ trợ ta bổ sung tài liệu cho hàm, việc này rất thuận lợi cho đối tác sử dụng các function do ta làm ra. Dựa vào những tài liệu này mà Partner có thể dễ dàng biết cách sử dụng.
- Ta có thể sử dụng 3 dấu nháy kép hoặc 3 dấu nháy đơn để viết tài liệu cho hàm. Tuy nhiên theo kinh nghiệm thì các bạn nên dùng 3 dấu nháy kép.
- Các ghi chú (tài liệu) phải được viết ở những dòng đầu tiên khi khai báo hàm

4.5. Viết tài liệu cho hàm

Ví dụ ta tạo 1 file

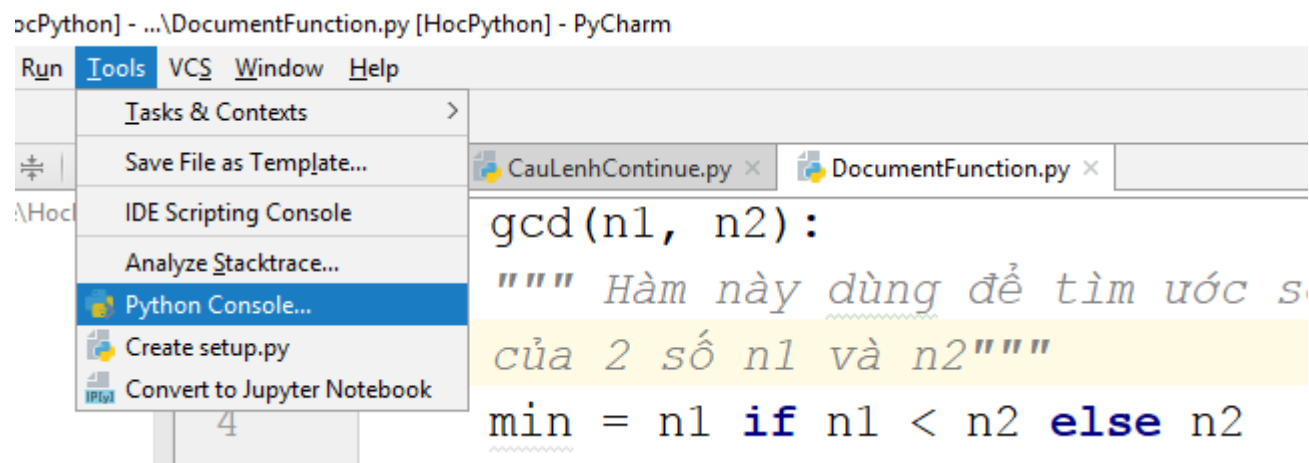
DocumentFunction.py

Có 2 hàm **gcd** và **ptb1**

```
1 def gcd(n1, n2):
2     """ Hàm này dùng để tìm ước số chung lớn nhất
3     của 2 số n1 và n2 """
4     min = n1 if n1 < n2 else n2
5     largest_factor = 1
6     for i in range(1, min + 1):
7         if n1 % i == 0 and n2 % i == 0:
8             largest_factor = i
9     return largest_factor
10 def ptb1(a,b):
11     """Giải phương trình bậc 1
12     ax+b=0"""
13     if a ==0 and b==0:
14         return "Vô số nghiệm"
15     elif a==0 and b!=0:
16         return "Vô nghiệm"
17     else:
18         return "x={0}".format(round(-b/a,2))
```

4.5. Viết tài liệu cho hàm

Ta chạy command line để xem cách lấy tài liệu cho các hàm trên. Ta vào menu tools/chọn Python Console...

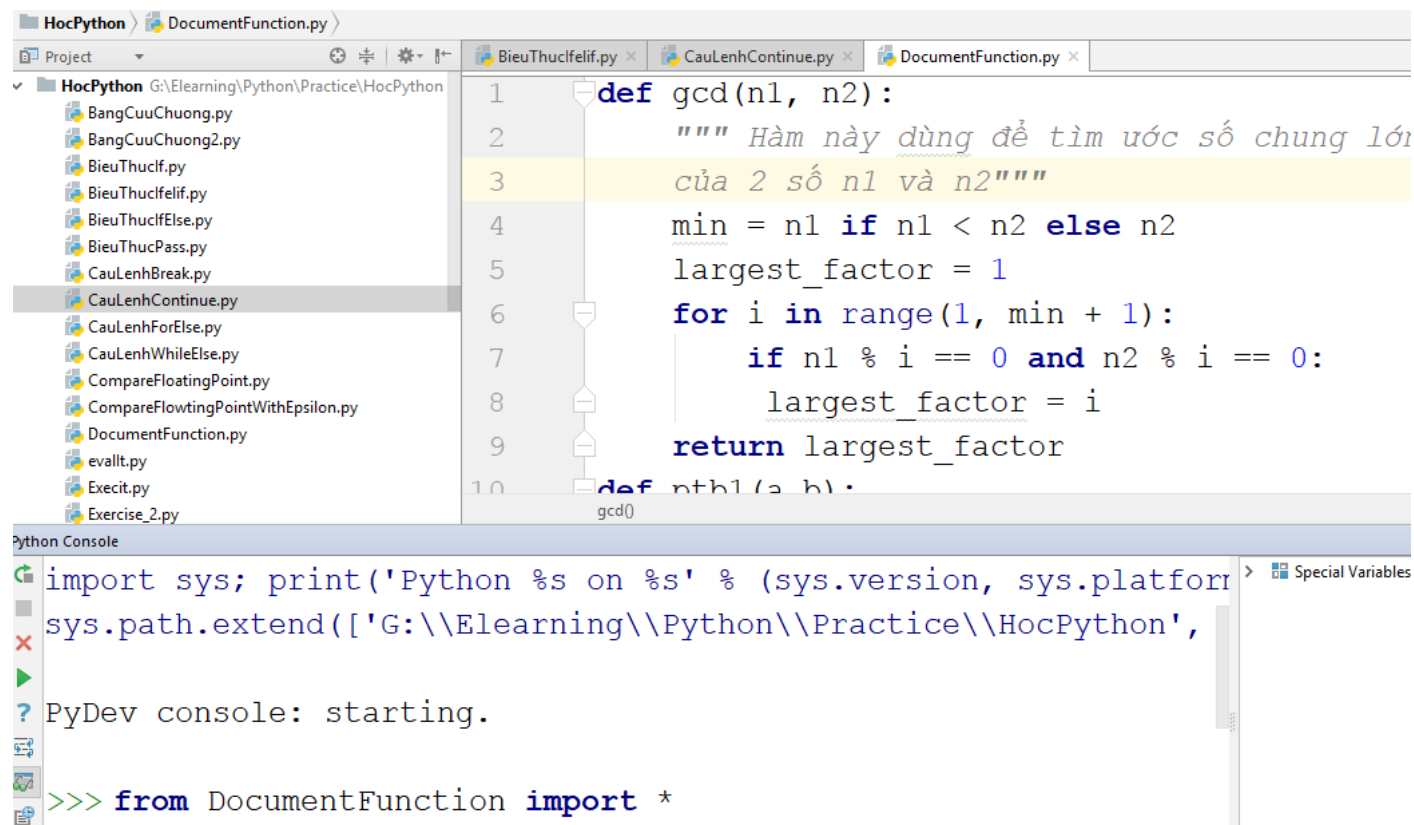


```
Python] - ...\\DocumentFunction.py [HocPython] - PyCharm
Run Tools VCS Window Help
Tasks & Contexts >
Save File as Template...
IDE Scripting Console
Analyze Stacktrace...
Python Console...
Create setup.py
Convert to Jupyter Notebook

gcd(n1, n2):
    """ Hàm này dùng để tìm ước số
    của 2 số n1 và n2 """
    min = n1 if n1 < n2 else n2
```

4.5. Viết tài liệu cho hàm

Ta chạy command line để xem cách lấy tài liệu cho các hàm trên. Ta vào menu tools/chọn Python Console...



The screenshot shows an IDE window with a project named 'HocPython'. The file explorer on the left lists several Python files, with 'CauLenhContinue.py' selected. The main editor displays the code for 'DocumentFunction.py', which defines a function 'gcd(n1, n2)'. The function has a docstring in Vietnamese: 'Hàm này dùng để tìm ước số chung lớn nhất của 2 số n1 và n2'. The code calculates the greatest common divisor (gcd) using a loop. Below the editor, the 'Python Console' window is open, showing the command 'from DocumentFunction import *' being entered.

```
def gcd(n1, n2):  
    """ Hàm này dùng để tìm ước số chung lớn  
    của 2 số n1 và n2 """  
    min = n1 if n1 < n2 else n2  
    largest_factor = 1  
    for i in range(1, min + 1):  
        if n1 % i == 0 and n2 % i == 0:  
            largest_factor = i  
    return largest_factor  
  
def nth1(a, b):  
    gcd()
```

```
import sys; print('Python %s on %s' % (sys.version, sys.platform))  
sys.path.extend(['G:\\Elearning\\Python\\Practice\\HocPython',  
PyDev console: starting.  
  
>>> from DocumentFunction import *
```

from DocumentFunction import *

4.5. Viết tài liệu cho hàm

Muốn xem tài liệu của hàm nào thì gõ : **help**(function name)

```
>>> from DocumentFunction import *
>>> help(gcd)
Help on function gcd in module DocumentFunction:

gcd(n1, n2)
    Hàm này dùng để tìm ước số chung lớn nhất
    của 2 số n1 và n2
```

```
>>> help(ptb1)
Help on function ptb1 in module DocumentFunction:

ptb1(a, b)
    Giải phương trình bậc 1
    ax+b=0
```

4.6. Global Variable

- Tất cả các biến khai báo trong hàm chỉ có phạm vi ảnh hưởng trong hàm, các biến này gọi là biến local. Khi thoát khỏi hàm thì các biến này không thể truy xuất được.

➤ Xem ví dụ sau:

```
1 g=5
2 def increment():
3     g=2
4     g=g+1
5     increment()
6     print(g)
```

Global variable

Local variable

Global variable

- Theo bạn thì chạy xong 6 dòng lệnh ở trên, giá trị g xuất ra màn hình bao nhiêu?

4.6. Global Variable

➤ Xem ví dụ 2 sau:

```
1 g=5
2 def increment():
3     global g
4     g=2
5     g=g+1
6     increment()
7     print(g)
```

- Theo bạn thì chạy xong 7 dòng lệnh ở trên, giá trị g xuất ra màn hình bao nhiêu?
- Global cho phép ta tham chiếu sử dụng được biến Global



4.6. Global Variable

➤ Xem ví dụ 3 sau:

```
1 g=5
2 def increment():
3     g=g+1
4     increment()
5     print(g)
```

➤ G dòng 3 Báo lỗi nha, vì g ở trong hàm không có lấy g ở ngoài (khai báo ở dòng 1)

4.7. Parameter mặc định

- Python cũng tương tự như C++, có hỗ trợ Parameter mặc định khi Khai báo hàm.
- Hàm print ta sử dụng cũng có các parameter mặc định

```
* def print(self, *args, sep=' ', end='\n', file=None): #  
    """  
    print(value, ..., sep=' ', end='\n', file=sys.stdout
```


4.7. Parameter mặc định

- Vậy nếu tự viết hàm thì ta sẽ định nghĩa các Parameter mặc định này như thế nào?

```
1  def SumRange (n, m=0) :  
2      sum=0  
3      for i in range (1, m+n, 1) :  
4          sum=i  
5      return sum  
6  
7  x1=SumRange (5)  
8  print ("x1=", x1)  
9  x2=SumRange (5, 1)  
10 print ("x2=", x2)
```

```
↓  
x1= 4  
x2= 5
```

4.8. Lambda Expression

- Python hỗ trợ kiểu khai báo hàm nặc danh thông qua Lambda expression:

lambda *parameterlist* : *expression*

lambda: là từ khóa

parameterlist: tập hợp các parameter mà ta muốn định nghĩa

expression: biểu thức đơn trong Python(ko nhập complex)

4.8. Lambda Expression

➤ Ví dụ ta định nghĩa 1 hàm

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

➤ Ta thấy đối số 1 là 1 hàm f nào đó

➤ Từ handle này ta 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 2 chấm là từ khóa lambda, đằng sau nó là số lượng các biến được khai báo trong handle (tính sau chữ f). Tức là nếu ta 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)
```

4.8. Lambda Expression

- Vì lambda expression không nhận cách viết complex, do đó nếu muốn complex thì ta nên định nghĩa các hàm độc lập rồi truyền vào cho Lambda expression. Ví dụ:
- Ở bên ta có 4 hàm, trong đó soChan, sole, SoNguyenTo sẽ được gọi vào handle

```
1  def handle(f,x):
2      return f(x)
3  def soChan(x):
4      return x%2==0
5  def sole(x):
6      return x%2==1
7  def soNguyenTo(x):
8      dem=0
9      for i in range(1,x+1):
10         if x % i is 0:
11             dem=dem+1
12     return dem==2
```

4.8. Lambda Expression

Một số Cách sử dụng:

```
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)
```



True
True
False
True
True
False
False
16

4.9. Giới thiệu về hàm đệ qui

- Đệ qui là cách mà hàm tự gọi lại chính nó, trong nhiều trường hợp đệ qui giúp ta giải quyết những bài toán học búa và theo “tự nhiên”. Tuy nhiên đệ qui 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 qui. Do đó có những bài toán Khử đệ qui thì ta nên nghĩ về các vòng lặp để khử.
- Một vài ví dụ kinh điển về đệ qui như tính giai thừa, tính dãy số Fibonacci.
- $N! = N * (N-1)!$ ➔ Đệ qui: Nếu biết được $(N-1)!$ Thì sẽ tính được $N!$
- $F_1=1, F_2=1, F_N=F_{N-1}+F_{N-2}$

4.9. Giới thiệu về hàm đệ qui

- Khi quyết định giải quyết bài toán theo Đệ Qui thì điều quan trọng phải nghĩ tới đó là: **Điểm dừng của bài toán là gì? + Biết được qui luật thực hiện của bài toán?** Nếu không tìm ra được 2 dấu hiệu này thì không thể giải quyết bài toán bằng cách dùng đệ qui.

4.9. Giới thiệu về hàm đệ qui

- Ví dụ ta thử phân tích bài giai thừa theo cách đệ qui?

$$n! = n \cdot (n-1) \cdot (n-2) \cdot (n-3) \cdots 3 \cdot 2 \cdot 1$$

- Viết lại dạng phương trình Đệ Qui có điều kiện:

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

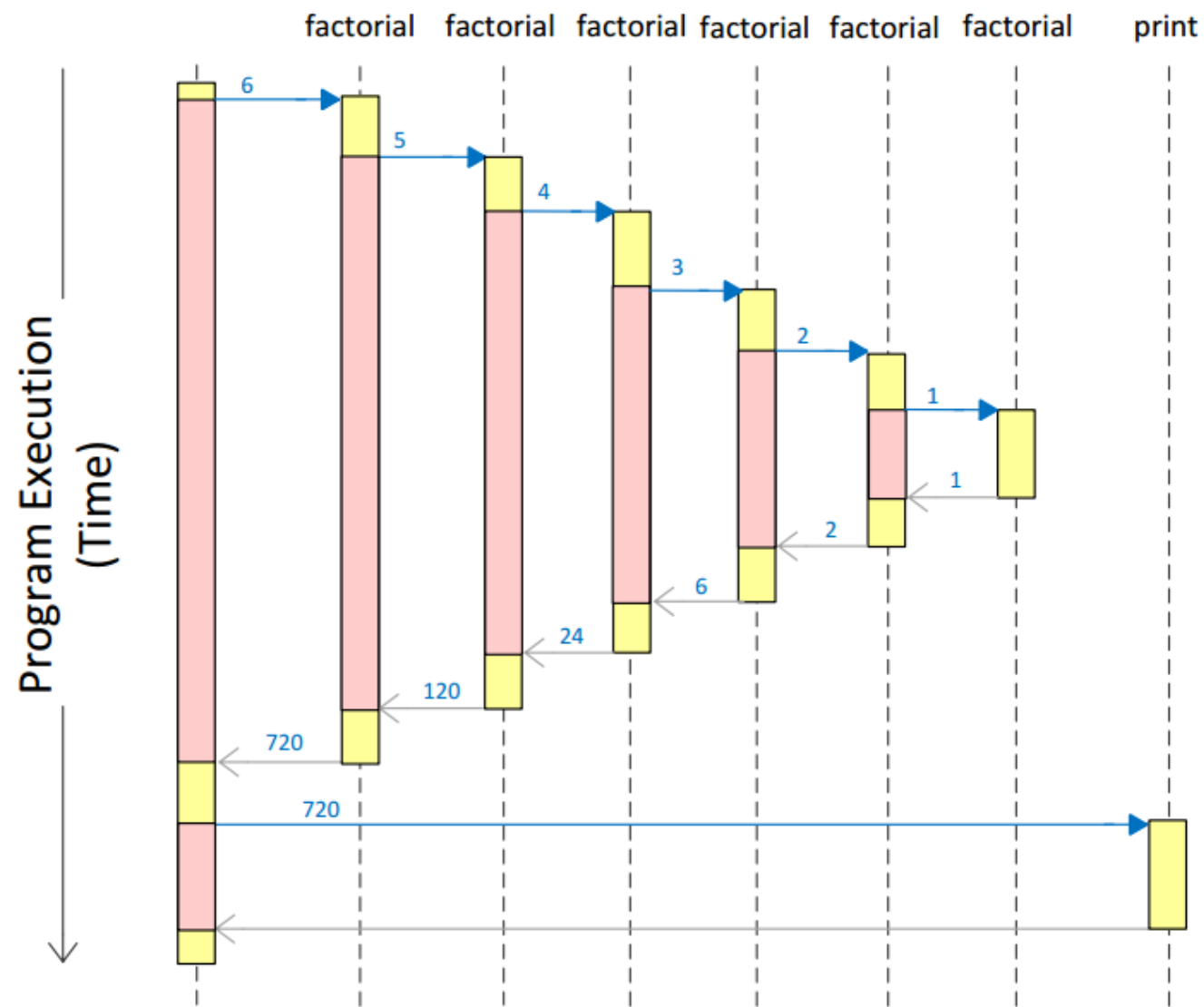
- Điểm dừng là khi $n=0$, quy luật là nếu biết $(n-1)!$ Thì tính được $N!$, vì $N! = N \cdot (N-1)!$

4.9. Giới thiệu về hàm đệ qui

```
def factorial(n):  
    """  
    Hàm tính n!  
    Trả về giai thừa của n  
    """  
    if n == 0:  
        return 1  
    else:  
        return n * factorial(n - 1)  
  
kq=factorial(6)  
print(kq)
```

```
factorial(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
```

4.9. Giới thiệu về hàm đệ qui





Nội dung bài học

4.10. một số hàm quan trọng thường dùng

4.10.1. Các hàm toán học

4.10.2. round

4.10.3. Time

4.10.4. Random

4.10.5. exit

4.10.6. eval

4.10.1. Các hàm toán học

Một số hàm ta phải sử dụng từ các thư viện có sẵn, như các hàm tính căn bậc 2, lũy thừa, log... Bài giảng này ta sẽ tập sử dụng:

- Sqrt-Căn bậc 2
- Pow – lũy thừa
- Log-> $\log(x)=\log_e x=\ln x$
- Log10- Logarit cơ số 10 của x, $\log_{10}(x)=\log_{10} x$
- Exp-tính e^x
- Degrees-Đổi radian ra độ
- Radians- Tính radian $180/PI*x$
- Fabs- tính giá trị tuyệt đối

4.10.1. Các hàm toán học

Ví dụ:

```
from math import *  
print('sqrt(25)=', sqrt(25))  
print('pow(5,3)=', pow(5,3))  
print('log(2)=', log(2))  
print('log10(100)=', log10(100))  
print('exp(2)=', exp(2))  
print(degrees(0.5235987755982988))  
print(radians(30))
```



```
sqrt(25)= 5.0  
pow(5,3)= 125.0  
log(2)= 0.6931471805599453  
log10(100)= 2.0  
exp(2)= 7.38905609893065  
29.999999999999996  
0.5235987755982988
```

4.10.1. Các hàm toán học

Bài học này trình bày cách sử dụng các hàm Lượng giác trong Python

-Từ một góc ta đưa về Radian sau đó đẩy vào các hàm lượng giác:

- Sin
- Cos
- tan



4.10.2. round

Trong quá trình tính toán ta sẽ nhận được các số lẻ, các số này cần được làm tròn để cùng thỏa mãn yêu cầu nào đó. Hàm round được Python hỗ trợ như sau:

- `round(số gốc, đơn vị làm tròn)`

```
a=3
```

```
b=11
```

```
c=b/a
```

```
print(c) → 3.6666666666666665
```

```
print(round(c, 2)) → 3.67
```



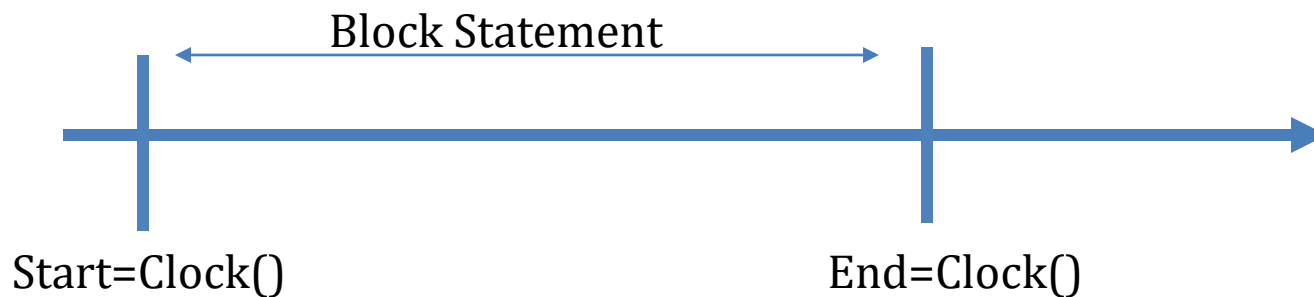
4.10.3. Time

Module time trong Python cung cấp rất nhiều hàm hữu ích, bài học giới thiệu 2 hàm **clock** và **sleep**.

1. Clock
2. Sleep

Clock

Clock tùy thuộc vào hệ điều hành mà cách thức xử lý khác nhau. Với Windows thì Clock trả về số giây khi ta gọi hàm clock. Để tính thời gian thực hiện một chương trình ta có thể căn 2 đầu để trừ ra số giây



Thời gian thực thi = End - Start



Clock

```
from time import clock
print("Enter your name: ", end="")
start_time = clock()
name = input()
elapsed = clock() - start_time
print(name, "it took you", elapsed, "seconds to respond")
```

Sleep

Sleep giúp ta tạm dừng quá trình chạy trong một đơn vị thời gian nào đó. Thay vì chương trình chạy một mạch, khi gặp sleep nó sẽ tạm dừng lại

```
from time import sleep
for count in range(10, -1, -1): # Range 10, 9, 8, ..., 0
    print(count) # Display the count
    sleep(1) # Suspend execution for 1 second
```



4.10.4. Random

Random là một trong những hàm khá hữu dụng trong việc viết Games, giả lập dữ liệu, thống kê.

`randrange(x,y)` → lấy số ngẫu nhiên $\geq x$ và $< y$

4.10.5. exit

Hàm exit dùng để thoát phần mềm

```
while True:
    s=input("Tên bạn:")
    print(s)
    hoi=input("Tiếp hay không? (c/k) :")
    if hoi=="k":
        exit()
print("BYE!")
```

4.10.6. eval

Hàm eval rất lợi hại, nó có thể tự tính toán chuỗi phép toán:

```
from math import sin
x=eval("1+2+5+sin(30)")
print(x)
```

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



THANK YOU

028 37244555 www.uel.edu.vn

ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC KINH TẾ - LUẬT

Số 669, đường Quốc lộ 1, khu phố 3, phường Linh Xuân,
quận Thủ Đức, Thành phố Hồ Chí Minh.