LẬP TRÌNH VỚI PYTHON

(Programming With Python)

Th.S Nguyễn Hoàng Thành

Email: thanhnh@ptithcm.edu.vn

Tel: 0909 682 711

HÀM

1. ĐẶT VẤN ĐỀ
2. HÀM LÀ GÌ
3. XÂY DỰNG HÀM
4. SỬ DỤNG HÀM
5. TRUYỀN THAM SỐ

1. ĐẶT VẤN ĐỀ

Bài toán 1

- Xây dựng chương trình thực hiện tính tổ hợp chập k của n: C_n^k
- Biết rằng:

$$C_n^k = \frac{n!}{k!(n-k)!} (1)$$

- Ví dụ:

$$C_4^2 = \frac{4!}{2!\,(4-2)!} = \frac{24}{2x2} = 6$$

Bài toán 1 (tt)

❖ **Thuật toán:**

- ❑ **Bước 1**: Nhập n và k từ bàn phím thỏa điều kiện $n \geq k > 0$
- ❑ **Bước 2**: Tính n giai thừa = 1 * 2 * 3 * … * n
- ❑ **Bước 3**: Tính k giai thừa = 1 * 2 * 3 * … * k
- ❑ **Bước 4**: Tính (n - k) giai thừa = 1 * 2 * 3 * … * (n-k)
- ❑ **Bước 5**: Giá trị tổ hợp chập k của n được tính bằng công thức (1) và hiển thị kết quả ra màn hình
- ❑ **Bước 6**: Kết thúc

Bài toán 1 (tt)

Chương trình 1.1:

```
while(True):
    n = int(input("Nhập n:"))
    k = int(input("Nhập k:"))
    if n >= k and k >0:
        break

giai_thua_n = 1
for i in range(1, n+1):
    giai_thua_n = giai_thua_n * i

giai_thua_k = 1
for i in range(1, k+1):
    giai_thua_k = giai_thua_k * i

giai_thua_nk = 1
for i in range(1, n-k+1):
    giai_thua_nk = giai_thua_nk * i

cnk = giai_thua_n/(giai_thua_k*giai_thua_nk)
print("To hop chap {0} cua {1} la {2}".format(k,n,cnk))
```

```
Nhập n:2
Nhập k:5
Nhập n:5
Nhập k:2
To hop chap 2 cua 5 la 10.0
```

Bài toán 1 (tt)

- Nhận xét:
  - Có một số mã chương trình lặp đi lặp lại
  - Có thể tái sử dụng lại bằng cách dùng hàm

Bài toán 1 (tt)

Chương trình 1.2:

```
while(True):
    n = int(input("Nhập n:"))
    k = int(input("Nhập k:"))
    if n >= k and k >0:
        break

def giai_thua(n):
    giai_thua = 1
    for i in range(1, n + 1):
        giai_thua = giai_thua * i
    return giai_thua

giai_thua_n = giai_thua(n)
giai_thua_k = giai_thua(k)
giai_thua_nk = giai_thua(n-k)

cnk = giai_thua_n/(giai_thua_k*giai_thua_nk)
print("To hop chap {0} cua {1} la {2}".format(k,n,cnk))
```

Bài toán 1 (tt)

- Chương trình 1.2 ngắn gọn hơn Chương trình 1.1

2. HÀM

Định nghĩa Hàm

- A function is a **block of organized**, **reusable code** that is used to perform a single, related action. Functions provides better modularity for your application and a high degree of code reusing.

Định nghĩa Hàm

❖ Hàm là một chương trình con gồm một tập các câu lệnh giải quyết một vấn đề cụ thể từ các tham số, gia trị truyền vào hàm sau khi thực hiện các câu lệnh hàm kết thúc có hoặc không trả về một giá trị cụ thể.

- Các hàm là cấu trúc chương trình cơ bản nhất mà Python cung cấp để tối đa hóa việc **sử dụng lại mã** và **giảm thiểu sự dư thừa mã**.
- Hàm cũng giúp chia các hệ thống phức tạp thành các phần có thể quản lý được.

Định nghĩa Hàm (tt)

- Một số hàm thường sử dụng trong Python:
  - print()
  - sum()
  - len()
  - input()

Table 16-1. Function-related statements and expressions

Câu lệnh	Ví dụ
call	myfunc('spam', 'eggs', meat=ham)
def return	def adder(a, b=1, *c): return a + b + c[0]
global	def changer(): global x; x = 'new'
nonlocal	def changer(): nonlocal x; x = 'new'
yield	def squares(x): for i in range(x): yield i ** 2
lambda	funcs = [lambda x: x**2, lambda x: x*3]

Tại sao sử dụng Hàm?

- Tối đa hóa việc sử dụng lại mã và giảm thiểu sự dư thừa: Hàm cho phép chúng ta nhóm và tổng quát hóa mã để sử dụng tùy ý nhiều lần sau này.
- Phân tách thủ tục: Các hàm cũng cung cấp một công cụ để chia hệ thống thành các phần có vai trò được xác định rõ.

3. XÂY DỰNG HÀM

3.1 Cách xây dựng Hàm

def *<tên hàm>* (*[<dsts>]*) :

<khối lệnh>

[**return** *<giá trị trả về>*]

❖ **Với:**

- ***<tên hàm>*: Do người dùng đặt theo quy tắc đặt tên**
- ***<dsts>*: Danh sách các tham số truyền vào làm giá trị đầu vào cho hàm để giải quyết công việc, <dsts> có thể có hoặc không, nếu có nhiều tham số cách nhau bởi dấu phẩy.**
- ***<giá trị trả về>:* Là giá trị trả về của hàm sau khi hoàn thành công việc nếu có lệnh return.**

```
def functionname( parameters ):
"function_docstring" function_suite return [expression]
def functionname( parameters ):
    "function_docstring"
    function_suite
    return [expression]
```

Các quy tắc để xác định một hàm trong Python

- Các khối hàm bắt đầu bằng từ khóa ***def*** theo sau là ***tên hàm*** và dấu ngoặc đơn (( )).
- Mọi tham số hoặc đối số đầu vào phải được đặt trong các dấu ngoặc đơn này. Bạn cũng có thể xác định các tham số bên trong các dấu ngoặc đơn này.
- Câu lệnh đầu tiên của một hàm có thể là một câu lệnh tùy chọn - chuỗi tài liệu của hàm hoặc chuỗi tài liệu.
- Khối mã trong mọi chức năng bắt đầu bằng **dấu hai chấm** (:) và được **thụt vào**.
- Câu lệnh return [biểu thức] thoát khỏi một hàm, tùy chọn trả lại một biểu thức cho người gọi. Câu lệnh trả về không có đối số cũng giống như câu lệnh trả về Không có.

3.2 Ví dụ 2

- Xây dựng hàm tính tổng hai số với tham số truyền vào hàm là hai số a và b, giá trị trả về của hàm là tổng của a và b.

```
def tong2so(a, b):
    c = a + b
    return c

print(tong2so(5, 7))
```

3.3 Ví dụ 3

- Xây dựng hàm kiểm tra tham số đầu vào có phải là số nguyên tố hay không. Giá trị trả về là True nếu tham số đầu vào là số nguyên tố. Ngược lại trả về là False.

```
def is_prime(n):
    for i in range(2, n):
        if n % i ==0:
            return False
    return True

print(is_prime(5))
```

4. SỬ DỤNG HÀM

Sử dụng hàm

❖ Gọi hàm để thực thi các khối lệnh bên trong thân hàm theo cú pháp sau:

[<tên biến> =] <tên hàm>([<dsts>])

❖ **Trong đó:**

- ❑ ***<tên biến>***: Biến được dùng để lưu dữ liệu sau khi kết thúc hàm.
- ❑ ***<dsts>***: Danh sách các tham số truyền vào làm giá trị đầu vào cho hàm, <dsts> có thể có hoặc không tùy và hàm đã xây dựng, nếu có nhiều tham số thì mỗi tham số cách nhau bởi dấu phẩy.

Ví dụ 4:

- Xây dựng hàm tính diện tích hình chữ nhật với tham số đầu vào là độ lớn hai cạnh a và b, giá trị trả về của hàm là diện tích của hình chữ nhật. Chương trình cho phép nhập vào chiều dài và chiều rộng hình chữ nhật, sau đó gọi hàm tính diện tích và hiển thị kết quả ra màn hình.

Ví dụ 4 (tt)

```
def dien_tich_hcn(a, b):
    return  a*b
while True:
    a = float(input("Nhập chiều dài: "))
    if a >0:
        break
while True:
    b = float(input("Nhập chiều rộng: "))
    if b >0:
        break
dt = dien_tich_hcn(a, b)
print("Diện tích hình chữ nhật:", dt)
```

```
Nhập chiều dài: 10
Nhập chiều rộng: 5
Diện tích hình chữ nhật: 50.0
```

5. TRUYỀN THAM SỐ

5. Truyền tham số cho Hàm

- 5.1 Truyền tham số với giá trị bắt buộc (Required argument)
- 5.2 Truyền tham số mặc định (Default argument)
- 5.3 Tham số từ khóa (Keyword argument)
- 5.4 Tham số tùy ý (Variable-length arguments)

5.1 Truyền tham số với giá trị bắt buộc

- Tham số bắt buộc là các tham số yêu cầu chúng ta buộc phải truyền vào trong lời gọi hàm theo đúng thứ tự nếu không chương trình sẽ thông báo lỗi.
- Cần quan tâm đến thứ tự của các tham số trong lời gọi hàm và nên kết nối chính xác với phần định nghĩa hàm.

Ví dụ 5

- Viết hàm thực hiện hoán vị hai số a và b

```
def hoan_vi(a, b):
    return b, a
a1 = 2
a2 = 5
print("a1 = {0}, a2 ={1}".format(a1, a2))
a1, a2 = hoan_vi(a1, a2)
print("a1 = {0}, a2 ={1}".format(a1, a2))
```

```
a1 = 2, a2 =5
a1 = 5, a2 =2
```

5.2 Truyền tham số mặc định

- Tham số là dữ liệu đầu vào của hàm, trong nhiều trường hợp người dùng không truyền tham số trong lời gọi hàm, điều này dẫn đến chương trình bị lỗi.
- Nhằm đảm bảo một hàm luôn được thực thi, chúng ta có thể thiết lập giá trị mặc định cho các tham số đầu vào trong quá trình xây dựng hàm.

5.2 Truyền tham số mặc định (tt)

Cú pháp

def *<tên hàm> ([<tham số 1 = giá trị 1>, <tham số 2> = <giá trị 2>,...]):*

<khối lệnh>

[return <giá trị trả về>]

Trong đó:

<tham số 1, tham số 2,...>: Là các tham số làm dữ liệu đầu vào của hàm do người dùng đặt theo quy tắc đặt tên.

<giá trị 1, giá trị 2,...>: Là các giá trị khởi tạo cho tên tham số. Nếu người dùng lời gọi hàm không truyền tham số thì hàm sẽ lấy các giá trị mặc định này để tính toán.

Ví dụ 6:

- Xây dựng hàm tính diện tích hình tròn với tham số đầu vào là bán kính đường tròn, mặc định bằng không.

```
def dien_tich_hinh_tron(r = 0):
    c = 3.14 * r * r
    return c

dt = dien_tich_hinh_tron()
print("Diện tích hình tròn bán kính mặc định là ",dt)
dt = dien_tich_hinh_tron(9)
print("Diện tích hình tròn bán kính = 9 là ",dt)
```

```
Diện tích hình tròn bán kính mặc định là  0.0
Diện tích hình tròn bán kính = 9 là  254.34
```

Ví dụ 7

- Xây dựng hàm tính tổng ước số dương của một số nguyên với đối số mặc định là 1

```
def tong_uoc(a = 1):
    if a < 0: a = -1*a
    tong_uoc_a = 0
    for uoc in range(a, a+1):
        if a % uoc == 0:
            tong_uoc_a = tong_uoc_a + uoc
    return tong_uoc_a

print("Tong uoc a với tham số mặc định là:",tong_uoc())
print("Tong uoc số dương của 6 là:",tong_uoc(6))
```

```
Tong uoc a với tham số mặc định là: 1
Tong uoc số dương của 6 là: 6
```

5.3 Tham số từ khóa

❖ Không cần nhớ vị trí của tham số mà chỉ cần nhớ tên tham số và truyền theo cú pháp sau:

[<tên biến> =] <tên hàm>([tham số 1=<giá trị 1>, tham số 2 = <giá trị 2>,...])

❖ Trong đó:

- ❑ *<tham số 1, 2, ...>*: là tham số chứa giá trị đầu vào cho hàm.
- ❑ *<giá trị 1, 2, ...>*: là danh sách các giá trị tương ứng với tên tham số trong hàm, không cần quan tâm đến vị trí của chúng.

Ví dụ 8:

- Gọi hàm tính diện tích hình chữ nhật và truyền tham số theo kiểu từ khóa.

```
def dien_tich_hcn(a, b):
    return a*b
print("Diện tích hình chữ nhật (5x25):", dien_tich_hcn(a = 5, b = 25))
print("Diện tích hình chữ nhật (3x5):", dien_tich_hcn(b = 5, a = 3))
```

```
Diện tích hình chữ nhật (5x25): 125
Diện tích hình chữ nhật (3x5): 15
```

5.4 Tham số tùy ý

- Người dùng chưa rõ có bao nhiêu tham số truyền vào hàm.
- Python hỗ trợ truyền tham số tùy ý bằng cách dùng dấu hoa thị * đặt trước tên tham số để biểu thị kiểu tham số này.

The Anonymous Functions

Syntax:

lambda [arg1 [,arg2,.....argn]]:expression

Example: Following is the example to show how lambda form of function works:

```
sum = lambda arg1, arg2: arg1 + arg2;

print("Value of total : ", sum( 10, 20 ))

print("Value of total : ", sum( 20, 20 ))
```

This would produce following result:

```
Value of total : 30

Value of total : 40
```

Ví dụ 9

- Xây dựng hàm tìm số lớn nhất của n số truyền vào hàm với n là tùy ý.

```
def max(*tham_so_tuy_y):
    if len(tham_so_tuy_y)==0:
        return
    else:
        max = tham_so_tuy_y[0]
        for x in tham_so_tuy_y:
            if max < x:
                max = x
        return max
print("số lớn nhất của 2 và 3: ",max(2, 3))
print("số lớn nhất của 12, -3 và 3: ",max(12, -3, 6))
```

```
số lớn nhất của 2 và 3:  3
số lớn nhất của 12, -3 và 3:  12
```

6. Đệ quy

- Đệ quy là một kỹ thuật lập trình mà cho phép một hàm gọi chính nó trong quá trình thực thi.
- Đệ quy giúp cho việc thiết kế và viết mã trở nên đơn giản hơn bằng cách giải quyết các vấn đề bằng cách chia nhỏ thành các vấn đề nhỏ hơn và giải quyết chúng riêng biệt.
- Cần phải xác định một hàm và định nghĩa cách hàm đó sẽ gọi chính nó.
- Phải đặt một điều kiện dừng để tránh việc hàm gọi chính nó mãi mãi, đưa đến một vòng lặp vô hạn.

Ví dụ:

```
>>> def countdown(n):
...     if n <= 0:
...         print("Blastoff!")
...     else:
...         print(n)
...         countdown(n-1)
...
>>> countdown(3)
3
2
1
Blastoff!
>>>
```

7. BIẾN

Biến toàn cục

- Một biến được khai báo **bên ngoài hàm** hoặc trong **phạm vi toàn cục** được gọi là biến toàn cục.
  - Điều này có nghĩa, biến toàn cục có thể được truy cập bên trong hoặc bên ngoài hàm.
- Biến toàn cục có tên gọi là **Global Variable**.

```
x = "Lập trình Python cơ bản"

def foo():
    print("Trong hàm:", x)

foo()
print("Ngoài hàm:", x)
```

- Các biến có thể được gán ở ba vị trí khác nhau, tương ứng với ba phạm vi khác nhau:
  - Nếu một biến được gán bên trong một def, thì nó là **cục bộ (local)** của hàm đó.
  - Nếu một biến được gán trong một def kèm theo, thì nó **không cục bộ (nonlocal)** đối với các hàm lồng nhau.
  - Nếu một biến được gán bên ngoài tất cả các def, nó là **toàn cục (global)** cho toàn bộ tệp.

Bài tập

- Trong toán học, ước số chung lớn nhất (ƯSCLN) hay ước chung lớn nhất (ƯCLN) của hai hay nhiều số nguyên là số nguyên dương lớn nhất là ước số chung của các số đó.
- Ví dụ, ước chung lớn nhất của 6 và 15 là 3 vì 6:3=2 và 15:3=5. Hãy viết một script tính ước số chung lớn nhất giữa hai số a và b.

MODULE

Python - Modules

Mô-đun giúp sắp xếp hợp lý mã Python của mình.

Nhóm mã liên quan vào một mô-đun làm cho mã dễ hiểu và dễ sử dụng hơn.

Mô-đun là một đối tượng Python với các thuộc tính được đặt tên tùy ý mà bạn có thể liên kết và tham chiếu.

Đơn giản, một mô-đun là một tệp bao gồm mã Python. Một mô-đun có thể định nghĩa các hàm, lớp và biến. Một mô-đun cũng có thể bao gồm mã có thể chạy được.

Example:

The Python code for a module named aname normally resides in a file named aname.py. Here's an example of a simple module, hello.py

```
def print_func( par ):
        print("Hello : ", par)
        return
```

The *import* Statement:

Bạn có thể sử dụng bất kỳ tệp nguồn Python nào làm mô-đun bằng cách thực thi câu lệnh nhập trong một số tệp nguồn Python khác. nhập khẩu có cú pháp sau:

```
import module1[, module2[,... moduleN]
```

Khi trình thông dịch gặp câu lệnh nhập, nó sẽ nhập mô-đun nếu mô-đun có trong đường dẫn tìm kiếm. Đường dẫn tìm kiếm là danh sách các thư mục mà trình thông dịch tìm kiếm trước khi nhập mô-đun.

The *from...import* * Statement:

Example:

```
import hello
hello.print_func("Zara")
```

This would produce following result:

```
Hello : Zara
```

Một mô-đun chỉ được tải một lần, bất kể số lần nó được nhập. Điều này ngăn quá trình thực thi mô-đun diễn ra lặp đi lặp lại nếu xảy ra nhiều lần nhập.

The *from...import* * Statement:

Cũng có thể nhập tất cả các tên từ một mô-đun vào không gian tên hiện tại bằng cách sử dụng câu lệnh nhập sau:

from modname import *

Điều này cung cấp một cách dễ dàng để nhập tất cả các mục từ một mô-đun vào không gian tên hiện tại; tuy nhiên, tuyên bố này nên được sử dụng một cách tiết kiệm.

The *from...import* * Statement:

Locating Modules:

Khi bạn nhập một mô-đun, trình thông dịch Python sẽ tìm kiếm mô-đun đó theo trình tự sau:

The current directory.

- Nếu không tìm thấy mô-đun, Python sẽ tìm kiếm từng thư mục trong biến shell PYTHONPATH.
- Nếu vẫn thất bại, Python sẽ kiểm tra đường dẫn mặc định. Trên UNIX, đường dẫn mặc định này thường là /usr/local/lib/python/.

Đường dẫn tìm kiếm mô-đun được lưu trữ trong sys mô-đun hệ thống dưới dạng biến sys.path. Biến sys.path chứa thư mục hiện tại, PYTHONPATH và mặc định phụ thuộc vào cài đặt.

FILES I/O

Python Files I/O

Printing to the Screen:

The simplest way to produce output is using the print statement where you can pass zero or more expressions, separated by commas. This function converts the expressions you pass it to a string and writes the result to standard output as follows:

```
print("Python is really a great language,", "isn't it?")
```

This would produce following result on your standard screen:

```
Python is really a great language, isn't it?
```

Reading Keyboard Input:

Python provides two built-in functions to read a line of text from standard input, which by default comes from the keyboard. These functions are:

- raw_input
- input

The raw_input Function:

The raw_input([prompt]) function reads one line from standard input and returns it as a string (removing the trailing newline):

```
str = raw_input("Enter your input: ");

print("Received input is : ", str)
```

This would prompt you to enter any string and it would display same string on the screen. When I typed **"Hello Python!"**, it output is like this:

```
Enter your input: Hello Python

Received input is : Hello Python
```

The input Function:

The *input([prompt])* function is equivalent to raw_input, except that it assumes the input is a valid Python expression and returns the evaluated result to you:

```
str = input("Enter your input: ");

print("Received input is : ", str)
```

This would produce following result against the entered input:

```
Enter your input: [x*5 for x in range(2,10,2)]

Received input is : [10, 20, 30, 40]
```

Opening and Closing Files:

Until now, you have been reading and writing to the standard input and output. Now we will see how to play with actual data files.

Python provides basic functions and methods necessary to manipulate files by default. You can do your most of the file manipulation using a file object.

The open Function:

Before you can read or write a file, you have to open it using Python's built-in open() function. This function creates a file object which would be utilized to call other support methods associated with it.

Syntax:

file object = open(file_name [, access_mode][, buffering])

Parameters detail:

- **file_name:** The file_name argument is a string value that contains the name of the file that you want to access.
- **access_mode:** The access_mode determines the mode in which the file has to be opened ie. read, write append etc. A complete list of possible values is given below in the table. This is optional parameter and the default file access mode is read (r)
- **buffering:** If the buffering value is set to 0, no buffering will take place. If the buffering value is 1, line buffering will be performed while accessing a file. If you specify the buffering value as an integer greater than 1, then buffering action will be performed with the indicated buffer size. If negative, the buffer size is the system default(default behavior).

A list of the different modes of opening a file:

Modes	Description
r	Opens a file for **reading only**. The file pointer is placed at the beginning of the file. This is the default mode.
rb	Opens a file for **reading only in binary forma**t. The file pointer is placed at the beginning of the file. This is the default mode.
r+	Opens a file for both **reading and writing**. The file pointer will be at the beginning of the file.
rb+	Opens a file for both **reading and writing in binary format**. The file pointer will be at the beginning of the file.

A list of the different modes of opening a file:

Modes	Description
w	Opens a file for **writing only.** Overwrites the file if the file exists. If the file does not exist, creates a new file for writing.
wb	Opens a file for **writing only in binary format.** Overwrites the file if the file exists. If the file does not exist, creates a new file for writing.
w+	Opens a file for both **writing and reading.** Overwrites the existing file if the file exists. If the file does not exist, creates a new file for reading and writing.
wb+	Opens a file for both **writing and reading in binary format**. Overwrites the existing file if the file exists. If the file does not exist, creates a new file for reading and writing.

A list of the different modes of opening a file:

Modes	Description
a	Opens a file for **appending**. The file pointer is at the end of the file if the file exists. That is, the file is in the append mode. If the file does not exist, it creates a new file for writing.
ab	Opens a file for **appending in binary format**. The file pointer is at the end of the file if the file exists. That is, the file is in the append mode. If the file does not exist, it creates a new file for writing.
a+	Opens a file for both **appending and reading.** The file pointer is at the end of the file if the file exists. The file opens in the append mode. If the file does not exist, it creates a new file for reading and writing.
ab+	Opens a file for both **appending and reading in binary format**. The file pointer is at the end of the file if the file exists. The file opens in the append mode. If the file does not exist, it creates a new file for reading and writing.

The file object attributes:

Once a file is opened and you have one file object, you can get various information related to that file.

Here is a list of all attributes related to file object:

Attribute	Description
file.closed	Returns true if file is closed, false otherwise.
file.mode	Returns access mode with which file was opened.
file.name	Returns name of the file.
file.softspace	Returns false if space explicitly required with print, true otherwise.

The file object attributes:

Example:

```
fo = open("foo.txt", "wb")

print("Name of the file: ", fo.name)

print("Closed or not : ", fo.closed)

print("Opening mode : ", fo.mode)

print("Softspace flag : ", fo.softspace)
```

This would produce following result:

```
Name of the file: foo.txt

Closed or not : False

Opening mode : wb

Softspace flag : 0
```

The close() Method:

The ***close()*** method of a file object flushes any unwritten information and closes the file object, after which no more writing can be done.

Python automatically closes a file when the reference object of a file is reassigned to another file. It is a good practice to use the **close()** method to close a file.

Syntax: fileObject.close();

Example:

```
fo = open("foo.txt", "wb")
print("Name of the file: ", fo.name)
fo.close()
```

This would produce following result:

```
Name of the file: foo.txt
```

Reading and Writing Files:

The file object provides a set of access methods to make our lives easier. We would see how to use read() and write() methods to read and write files.

The *write()* Method:

- The write() method writes any string to an open file. It is important to note that Python strings can have binary data and not just text.
- The write() method does not add a newline character ('\n') to the end of the string.

Syntax:

fileObject.write(string);

Reading and Writing Files:

Example:

```
fo = open("foo.txt", "wb")

fo.write( "Python is a great language.\r\nYeah it’s great!!\r\n");

fo.close()
```

The read() Method:

The read() method read a string from an open file. It is important to note that Python strings can have binary data and not just text.

Syntax:

```
fileObject.read([count]);
```

The read() Method:

Example:

```
fo = open("foo.txt", "r+")

str = fo.read(10);

print("Read String is : ", str)

fo.close()
```

This would produce following result:

```
Read String is : Python is
```

File Positions:

- The ***tell()*** method tells you the current position within the file in other words, the next read or write will occur at that many bytes from the beginning of the file:
- The ***seek(offset[, from])*** method changes the current file position. The offset argument indicates the number of bytes to be moved. The from argument specifies the reference position from where the bytes are to be moved.
- If ***from*** is set to 0, it means use the beginning of the file as the reference position and 1 means use the current position as the reference position and if it is set to 2 then the end of the file would be taken as the reference position.

File Positions:

Example:

```
fo = open("foo.txt", "r+")

str = fo.read(10);

print("Read String is : ", str)

position = fo.tell();

print("Current file position : ", position)

position = fo.seek(0, 0);

str = fo.read(10);

print("Again read String is : ", str)

fo.close()
```

This would produce following result:

```
Read String is : Python is

Current file position : 10

Again read String is : Python is
```

Renaming and Deleting Files:

Python **os** module provides methods that help you perform file-processing operations, such as renaming and deleting files.

To use this module you need to import it first and then you can all any related functions.

The rename() Method:

The rename() method takes two arguments, the current filename and the new filename.

Syntax:

os.rename(current_file_name, new_file_name)

Example:

```
import os

os.rename( "test1.txt", "test2.txt" )
```

The delete() Method:

You can use the delete() method to delete files by supplying the name of the file to be deleted as the argument.

Syntax:

```
os.remove(file_name)
```

Example:

```
import os

os.remove("test2.txt")
```

Directories in Python:

The mkdir() Method:

use the ***mkdir()*** method of the ***os*** module to create directories in the current directory. You need to supply an **argument** to this method, which contains the name of the directory to be created.

Syntax:

os.mkdir("newdir")

Example:

import os # Create a directory "test"

os.mkdir("test")

The chdir() Method:

You can use the chdir() method to change the current directory. The chdir() method takes an argument, which is the name of the directory that you want to make the current directory.

Syntax:

```
os.chdir("newdir")
```

Example:

```
import os

os.chdir("/home/newdir")
```

The getcwd() Method:

The getcwd() method displays the current working directory.

Syntax:

```
os.getcwd()
```

Example:

```
import os

os.getcwd()
```

The rmdir() Method:

The rmdir() method deletes the directory, which is passed as an argument in the method.

Before removing a directory, all the contents in it should be removed.

Syntax:

```
os.rmdir('dirname')
```

Example:

```
import os

os.rmdir( "/tmp/test" )
```

File & Directory Related Methods:

There are three important sources which provide a wide range of utility methods to handle and manipulate files & directories on Windows and Unix operating systems. They are as follows:

- File Object Methods: The file object provides functions to manipulate files.
- OS Object Methods.: This provides methods to process files as well as directories.