LẬP TRÌNH PYTHON

LÀM VIỆC VỚI TỆP VÀ QUẢN LÝ LỖI

NỘI DUNG

1. Làm việc với Tệp
2. Quản lý lỗi

LÀM VIỆC VỚI TỆP - FILE

- Để mở tệp chúng ta sử dụng hàm open(). Hàm open() nhận hai tham số là tên tệp và chế độ mở tệp, có 4 chế độ mở tệp khác nhau như sau:

"r" – Read – Đây là giá trị mặc định của hàm open(). Thực hiện mở tệp để đọc, có lỗi nếu tệp không tồn tại
"a" – Append – Mở một tệp để ghi thêm dữ liệu vào cuối tệp, tạo tệp mới nếu tệp chưa tồn tại
"w" - Write – Mở tệp để ghi, tạo tệp mới nếu tệp chưa tồn tại
"x" - Create – Tạo một tệp cụ thể, trả lại lỗi nếu tệp đã tồn tại

- Tham số xác định kiểu tệp

"t" - Text - Default value. Tệp văn bản
"b" - Binary – Tệp nhị phân (ví dụ như tệp hình ảnh)

LÀM VIỆC VỚI TỆP - FILE

- Ví dụ mở tệp:

```
f = open("demofile.txt")
```

- Câu lệnh trên tương đương với câu lệnh sau:

```
f = open("demofile.txt", "rt")
```

LÀM VIỆC VỚI TỆP - FILE

- Để đọc dữ liệu từ tệp, trước tiên bạn cần mở tệp để đọc thông qua hàm `open()`, hàm open sẽ trả lại một đối tượng tệp (file object), đối tượng tệp có phương thức `read()` để đọc nội dung của tệp.

```
f = open("demofile.txt", "r")
print(f.read())
```

demofile.txt

Nội dung tệp
Gồm 2 dòng

```
>>>Ná»™i dung tá»‡p
>>>Gá»“m 2 dÃ²ng
???
```

LÀM VIỆC VỚI TỆP - FILE

- Để đọc file với nội dung chứa dữ liệu Unicode (ví dụ utf-8), thêm tham số `encoding="utf-8"` khi mở tệp.

```
f = open("demofile.txt", "r", encoding="utf-8")
print(f.read())
```

>>> Nội dung tệp

>>> Gồm 2 dòng

demofile.txt

Nội dung tệp
Gồm 2 dòng

LÀM VIỆC VỚI TỆP - FILE

- Mặc định thì phương thức `read()` trả lại nội dung của cả tệp, tuy nhiên chúng ta có thể chỉ thị cho phương thức chỉ lấy ra một số ký tự nhất định.

```
f = open("demofile.txt", "r", encoding="utf-8")
print(f.read(5))
```

```
>>>Nội d
```

demofile.txt

Nội dung tệp
Gồm 2 dòng

LÀM VIỆC VỚI TỆP - FILE

- Để đọc từng dòng của tệp ta sử dụng phương thức `readline()`

```
f = open("demofile.txt", "r", encoding="utf-8")
print(f.readline()) >>>Nội dung tệp
print(f.readline()) >>>Gồm 2 dòng
```

- Về cơ bản, khi các thao tác làm việc với tệp hoàn tất chúng ta nên đóng tệp lại

```
f = open("demofile.txt", "r")
print(f.readline())
f.close()
```

demofile.txt

Nội dung tệp
Gồm 2 dòng

LÀM VIỆC VỚI TỆP - FILE

- Để ghi dữ liệu vào tệp thì việc trước tiên cần làm là mở tệp với chế độ ghi, có 2 chế độ cho việc ghi tệp là thêm dữ liệu vào cuối tệp và ghi đè dữ mới liệu vào tệp.

```
f = open("demofile.txt", "a",encoding ='utf-8')
f.write('Thêm 1 dòng nữa là 3')
f.close()
f =  open('demofile.txt','r',encoding = 'utf-8')
print(f.read())
```

demofile.txt

Nội dung tệp
Gồm 2 dòng
Thêm 1 dòng nữa là 3

LÀM VIỆC VỚI TỆP - FILE

- Ghi đè dữ liệu

```
f = open("demofile.txt", "w",encoding ='utf-8')
f.write('Dữ liệu đã bị ghi đè')
f.close()


f =  open('demofile.txt','r',encoding = 'utf-8')
print(f.read())
```

demofile.txt

Dữ liệu đã bị ghi đè

LÀM VIỆC VỚI TỆP - FILE

- Duyệt qua từng dòng với vòng lặp for

```
f = open('demofile.txt','w',encoding = 'utf-8')
f.write('Dòng 1\nDòng 2\nDòng 3\n')
f.close()

for line in open('demofile.txt','r',encoding='utf-8'):
    print(line,end='')
```

demofile.txt

Dòng 1
Dòng 2
Dòng 3

LÀM VIỆC VỚI TỆP - FILE

- Đọc tất cả các dòng thành danh sách

```
f = open('demofile.txt','r',encoding = 'utf-8')


#print(list(f))
print(f.readlines())
>>>['Dòng 1\n', 'Dòng 2\n', 'Dòng 3\n']
```

demofile.txt

Dòng 1
Dòng 2
Dòng 3

LÀM VIỆC VỚI TỆP - FILE

- Xóa tệp

Để xóa tệp, bạn cần thêm vào module os và gọi tới hàm `os.remove()`

```
import os

os.remove('demofile.txt')
```

- *Xóa thư mục*

Để xóa thư mục bạn dùng phương thức `os.rmdir()`

```
import os
os.rmdir("myfolder")
```

LÀM VIỆC VỚI TỆP - FILE

- Kiểm tra tệp đã tồn tại hay không

```
import os
if os.path.exists("demofile.txt"):
  os.remove("demofile.txt")
else:
  print("The file does not exist")
```

LÀM VIỆC VỚI TỆP - FILE

- Mẫu with open(...) as ...

```
with open('data.txt', 'r') as f:
    data = f.read()


with ('data.txt', 'w') as f:
    data = 'some data to be written to
the file'
    f.write(data)
```

LÀM VIỆC VỚI TỆP - FILE

- Làm việc với dữ liệu định dạng JSON

```
f = open('sample.json','w')
json.dump(x, f)
f = open('sample.json','r')
x = json.load(f)

print(x)
```

```
import json
x =  {
  "child1" : {
    "name" : "Emil",
    "year" : 2004
  },
  "child2" : {
    "name" : "Tobias",
    "year" : 2007
  },
  "child3" : {
    "name" : "Linus",
    "year" : 2011
  }
}
```

LÀM VIỆC VỚI TỆP - FILE

- Đọc ghi file với pickle
- pickle cũng hỗ trợ việc đọc và ghi đối tượng Python

```
import pickle
data = {
    'a': [1, 2.0, 3, 4+6j],
    'b': ("character string", b"byte string"),
    'c': {None, True, False}
}

with open('data.pickle', 'wb') as f:
    pickle.dump(data, f, pickle.HIGHEST_PROTOCOL)

with open('data.pickle', 'rb') as f:
    data = pickle.load(f)
    print(data)
```

QUẢN LÝ LỖI

Có hai loại lỗi cơ bản thường gặp phải khi viết chương trình là:

- Lỗi cú pháp (Syntax Errors)
- Ngoại lệ (Exceptions)

QUẢN LÝ LỖI

Lỗi cú pháp

- Lỗi cú pháp còn được gọi là lỗi phân tích là một loại lỗi thường gặp phải trong khi bạn đang học ngôn ngữ Python:

```
>>> while True print('Hello world')
  File “<stdin>” line 1
    while True print('Hello world')
                     ^

SyntaxError: invalid syntax
```

QUẢN LÝ LỖI

Ngoại lệ

- Ngay cả khi một câu lệnh đúng về cú pháp thì cũng có thể xảy ra lỗi khi thực thi. Lỗi được phát hiện khi thực thi được gọi là Ngoại lệ.

```
>>> 10 * (1/0)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ZeroDivisionError: division by zero
>>> 4 + spam*3
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'spam' is not defined
>>> '2' + 2
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: Can't convert 'int' object to str implicitly
```

QUẢN LÝ LỖI

Xử lý ngoại lệ

- Người lập trình cần xử lý các ngoại lệ gặp phải khi xây dựng các chương trình.
- Các lỗi ngoại lệ có thể được xử lý bằng lệnh try/except

```
>>> while True:
...     try:
...         x = int(input("Nhập vào một số nguyên: "))
...         break
...     except ValueError:
...         print("Bạn cần nhập vào một số nguyên")
...
```

QUẢN LÝ LỖI

Lệnh `try` hoạt động như sau:

- Đầu tiên, các lệnh nằm giữa `try` và `except` sẽ được thực thi
- Nếu không có ngoại lệ nào xảy ra, các lệnh trong khối except sẽ được bỏ qua, và các lệnh trong `try` sẽ được hoàn tất.
- Nếu có ngoại lệ xảy ra, các câu lệnh phía sau lệnh gây ra ngoại lệ sẽ không được thực thi. Khi đó nếu ngoại lệ trùng với loại ngoại lệ được liệt kê phía sau từ khóa `except` thì các lệnh trong khối excpte sẽ được thực thi, và chương trình thực thi các lệnh phía sau `try/except`
- Nếu ngoại lệ xảy ra mà không trùng với các loại ngoại lệ được liệt kê sau except, trình thực thi sẽ dừng lại và lỗi sẽ trả lại thông báo lỗi.

Một lệnh `try` có thể có nhiều hơn một mệnh đề except để thực hiện những xử lý riêng cho từng loại ngoại lệ. Tuy nhiên chỉ có nhiều nhất là một xử lý được thực thi tại một thời điểm.

QUẢN LÝ LỖI

Đẩy ra ngoại lệ

- Câu lệnh raise cho phép người lập trình đẩy ra một ngoại lệ. Ví dụ:

```
>>> raise NameError('HiThere')
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: HiThere

raise ValueError  # shorthand for 'raise
ValueError()'
```

- Sử dụng raise để in ra thông báo về ngoại lệ được xử lý

```
try:
    raise NameError('HiThere')
except NameError:
    print('An exception flew by!')
    raise
```

QUẢN LÝ LỖI

- Ngoại lệ do người dùng định nghĩa

```
class Error(Exception):
    """Base class for exceptions in this module."""
    pass

class InputError(Error):
    """Exception raised for errors in the input.

    Attributes:
        expression -- input expression in which the
error occurred
        message -- explanation of the error
    """

    def __init__(self, expression, message):
        self.expression = expression
        self.message = message
```

QUẢN LÝ LỖI

Clean-up Actions

- Các lệnh trong khối `finally` sẽ luôn được thực thi trong mọi hoàn cảnh.

```
>>> try:
...     raise KeyboardInterrupt
... finally:
...     print('Goodbye, world!')
...
Goodbye, world!
KeyboardInterrupt
Traceback (most recent call last):
  File "<stdin>", line 2, in <module>
```

Hết bài