AI VIET NAM – WARMUP 2025

M01W01 - Self-study (1)

Ngày 31 tháng 5 năm 2025

I. Lý thuyết

A. Các lỗi thường gặp

Trong quá trình lập trình Python, chúng ta thường xuyên gặp phải những lỗi không mong muốn. Hướng dẫn này sẽ giúp bạn hiểu rõ về các lỗi phổ biến nhất trong Python để nhanh chóng xác định và sửa lỗi, từ đó làm cho quá trình viết mã trở nên hiệu quả hơn!

1. Zero Division Error

Lỗi **ZeroDivisionError** là một trong những lỗi phổ biến trong lập trình Python, và nó xảy ra khi một phép chia có mẫu bằng không. Trong toán học, chia một số cho 0 là một phép toán không xác định và không có giá trị.

Khi chúng ta thực hiện một phép chia trong Python, nếu mẫu (số chia) có giá trị là 0, Python sẽ đưa ra một lỗi là `ZeroDivisionError`. Điều này thường xảy ra khi một biến hoặc biểu thức được sử dụng làm mẫu có giá trị bằng 0 trong quá trình thực thi chương trình.

Trong chương trình sau, thông báo `ZeroDivisionError` xuất hiện tại dòng thứ 6 khi thực hiện phép chia `c = a/b`, đây là lỗi chia cho 0 vì `a = 10, b = 0`.

```python
# Khai bao bien a va b
a = 10
b = 0

# Thuc hien phep chia
c = a / b

# In ket qua
print(c)
```

```
---------------------------------------------------------------------------
ZeroDivisionError                         Traceback (most recent call last)
<ipython-input-22-1548ebc2e105> in <cell line: 6>()
      4
      5 # Thuc hien phep chia
----> 6 c = a / b
      7
      8 # In ket qua

ZeroDivisionError: division by zero
```

Để tránh lỗi này, chúng ta cần kiểm tra giá trị của mẫu trước khi thực hiện phép chia, đặc biệt là khi giá trị của mẫu có thể trở thành 0 trong một số trường hợp. Chúng ta nên sử dụng các điều kiện kiểm tra để đảm bảo rằng chương trình không thực hiện phép chia với mẫu bằng 0, điều này sẽ giúp tăng tính ổn định của chương trình Python.

```python
# Khai bao bien a va b
a = 10
b = 0
```

```
# Thuc hien phep chia neu b khong bang 0
if b != 0:
    c = a / b
    # In ket qua
    print(c)
else:
    print("Cannot divide by zero")

>>Output: Cannot divide by zero
```

2. Name Error

Lỗi **NameError** xuất hiện khi một biến hoặc tên (tên hàm, tên thư viện, ...) không được định nghĩa trong ngữ cảnh hiện tại của chương trình. `NameError` thường xảy ra khi sử dụng một tên hàm, tên biến không tồn tại hoặc đã bị xóa từ bộ nhớ.

Ví dụ, nếu chúng ta cố gắng sử dụng một biến mà chưa được khai báo hoặc một tên hàm mà chưa được định nghĩa trước đó, Python sẽ thông báo lỗi `NameError`. Điều này có thể xảy ra khi chúng ta gặp phải các lỗi chính tả trong tên biến hoặc khi biến đó không tồn tại trong phạm vi hiện tại của chương trình.

Để khắc phục lỗi `NameError`, chúng ta cần đảm bảo rằng tên biến hoặc hàm chúng ta đang sử dụng đã được khai báo hoặc được định nghĩa trong phạm vi mà chúng ta đang làm việc. Kiểm tra cẩn thận các tên và chắc chắn rằng chúng đã được định nghĩa trước khi sử dụng sẽ giúp tránh được lỗi này.

Ví dụ 1:

```
# Khai bao bien a = 5
a = 5

# Thuc hien a + b, sau do luu vao
    c
c = a + b

# In gia tri c
print(c)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-1-0e3ead398d4d> in <cell line: 5>()
      3
      4 # Thực hiện a + b, sau đó lưu vào biến c
----> 5 c = a + b
      6
      7 # In giá trị c

NameError: name 'b' is not defined
```

Trong ví dụ trên, biến b không được định nghĩa trước khi sử dụng trong phép toán `a + b`. Python không nhận ra `b` là một biến đã được khai báo, do đó nó thông báo lỗi `NameError`.

Ví dụ 2:

```
# Khai bao bien a = 5
a = 5

# In gia tri a
Print(a)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-2-ef11efcc4e0d> in <cell line: 5>()
      3
      4 # In giá trị a
----> 5 Print(a)

NameError: name 'Print' is not defined
```

Trong ví dụ trên, hàm in được gọi bằng tên `Print` thay vì `print`. Python phân biệt chữ hoa và chữ thường, nên với `Print`, Python sẽ không tìm thấy hàm in và thông báo lỗi `NameError`. Để sửa, chúng ta sẽ viết là `print(a)`.

Ví dụ 3:

```
def a_function(x):
    a_variable = 4
    result = x*a_variable

    return result

print(a_variable)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-3-c39c0aa9070d> in <cell line: 7>()
      5     return result
      6
----> 7 print(a_variable)

NameError: name 'a_variable' is not defined
```

Trong ví dụ 3, tên biến `a_variable` được khai báo trong hàm `a_function`, nên nó chỉ tồn tại trong phạm vi của hàm đó. Khi chúng ta cố gắng in giá trị của `a_variable` bên ngoài hàm, Python sẽ không nhận ra nó và thông báo lỗi `NameError`. Để sửa lỗi này, chúng ta in giá trị `a_variable` bên trong hàm `a_function`.

3. Syntax Error

Lỗi **SyntaxError** xảy ra khi trình thông dịch Python phát hiện ra một cấu trúc ngôn ngữ không hợp lệ trong mã nguồn của chúng ta. Điều này có thể là do sử dụng cú pháp không đúng, như việc thiếu hoặc dư thừa dấu ngoặc đơn, dấu ngoặc vuông, dấu nháy, hoặc sử dụng từ khoá không đúng cách.

Ví dụ, nếu chúng ta đặt một dấu hai chấm `":"` sau một câu lệnh mà không có cấu trúc sử dụng dấu hai chấm, Python sẽ báo lỗi `SyntaxError`. Cũng có thể là do việc sử dụng khoảng trắng không đúng, thiếu dấu chấm phẩy, hoặc bất kỳ lỗi cú pháp nào khác mà Python không thể hiểu.

Để sửa lỗi `SyntaxError`, chúng ta cần kiểm tra cú pháp trong chương trình của chúng ta, đảm bảo rằng mọi cấu trúc ngôn ngữ được viết đúng và đúng định dạng. Một cách tiếp cận khác là xem xét thông báo lỗi và kiểm tra các đoạn mã xung quanh dòng code bị lỗi để xác định vị trí và sửa chúng.

Ví dụ 1:

```
# Khai bao bien chuoi s
s = 'Hello AIVIETNAM"

# In gia tri s
print(s)
```

```
  File "<ipython-input-4-3e2110290590>", line 2
    s = 'Hello AIVIETNAM"
        ^
SyntaxError: unterminated string literal (detected at line 2)
```

Trong ví dụ trên, khi khai báo biến chuỗi `s`, dấu nháy đơn bắt đầu bằng `"'"` nhưng kết thúc bằng dấu nháy kép `"""` , gây ra lỗi cú pháp. Để sửa lỗi này, chúng ta cần thêm một dấu nháy đơn ở cuối chuỗi: `s = 'Hello AIVIETNAM'`.

Ví dụ 2:

```
import math

number = 20.2
print(math.floor(number)
print(math.pi)
```

```
  File "<ipython-input-5-d922ea56cd8b>", line 4
    print(math.floor(number)
                            ^
SyntaxError: '(' was never closed
```

Ví dụ trên bị thiếu dấu đóng ngoặc đơn `")"` trong lệnh in `print(math.floor(number)`. Để sửa lỗi, chúng ta cần thêm một dấu đóng ngoặc đơn vào cuối lệnh in: `print(math.floor(number))`.

Ví dụ 3:

```
print "aivietnam.ai"

==================================
```

```
  File "<ipython-input-6-bdbaa7b60bf4>", line 1
    print "aivietnam.ai"
          ^
SyntaxError: Missing parentheses in call to 'print'. Did you mean print(...)?
```

Đoạn code trên xảy ra lỗi do viết sai cú pháp, thiếu cặp dấu `()` của hàm `print`, cách viết đúng: `print("aivietnam.ai")`.

```
number = 15
if number < 10:
    print("A small number")
else
    print("A large number")
```

```
  File "<ipython-input-7-a69fe018cc38>", line 4
    else
        ^
SyntaxError: expected ':'
```

Ví dụ trên thiếu dấu hai chấm `":"` sau câu lệnh `else`. Python yêu cầu một dấu hai chấm để đánh dấu kết thúc điều kiện. Để sửa lỗi, chúng ta cần thêm dấu hai chấm sau `else`.

4. Type Error

Lỗi **TypeError** xuất hiện khi một toán tử hoặc hàm được áp dụng cho một kiểu dữ liệu không phù hợp trong Python. Điều này có nghĩa là chúng ta đang cố gắng thực hiện một phép toán hoặc sử dụng một hàm trên một kiểu dữ liệu mà không hợp lý.

Ví dụ, nếu chúng ta cố gắng cộng hai chuỗi sử dụng phép toán `+`, nhưng một trong những toán hạng không phải là chuỗi, Python sẽ đưa ra lỗi `TypeError`. Tương tự, nếu chúng ta sử dụng một hàm yêu cầu đối số là một kiểu dữ liệu cụ thể, và chúng ta truyền một kiểu dữ liệu khác, cũng sẽ gây ra lỗi `TypeError`.

Để sửa lỗi `TypeError`, cần kiểm tra kiểu dữ liệu của các đối tượng tham gia phép toán hoặc được truyền vào hàm, và đảm bảo chúng tương thích với nhau. Chúng ta nên thực hiện chuyển đổi kiểu dữ liệu để đảm bảo tính hợp lý cho phép toán hoặc hàm.

Ví dụ 1:

```
# Khai bao bien chuoi s
s = 'AI'

# Khai bao bien n co kieu integer
n = 5

# Tinh gia tri c
c = s + n

# In gia tri c
print(c)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-8-66377ea73092> in <cell line: 8>()
      6
      7 # Tính giá trị c
----> 8 c = s + n
      9
     10 # In giá trị c

TypeError: can only concatenate str (not "int") to str
```

Ví dụ 1 thực hiện cộng một chuỗi `s` với một số nguyên `n`. Phép toán cộng này yêu cầu cả hai toán hạng có cùng kiểu dữ liệu. Để sửa lỗi, chúng ta có thể chuyển đổi số nguyên `n` thành chuỗi trước khi thực hiện phép cộng: `c = s + str(n)`.

Ví dụ 2:

```
a_number = 5
a_string = 'value '
result = a_string + a_number

print(result)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-10-52ca11d0ed4f> in <cell line: 3>()
      1 a_number = 5
      2 a_string = 'value '
----> 3 result = a_string + a_number
      4
      5 print(result)

TypeError: can only concatenate str (not "int") to str
```

Chương trình trên thực hiện cộng một chuỗi `a_string` với một số nguyên `a_number`, điều này cũng gây ra lỗi `TypeError`. Để sửa lỗi này cần chuyển đổi số nguyên thành chuỗi trước khi thực hiện phép cộng: `result = a_string + str(a_number)`.

5. Indentation Error

Lỗi **IndentationError** xuất hiện khi có sự không nhất quán trong việc thụt lề (indentation) của mã nguồn Python. Chúng ta cần biết rằng, Python sử dụng thụt lề để xác định phạm vi của các khối mã, thay vì sử dụng dấu ngoặc như nhiều ngôn ngữ khác. Do đó, việc giữ cho thụt lề đồng nhất là rất quan trọng.

Ví dụ, nếu chúng ta có một lệnh trong một khối mã nhưng các dòng trong khối đó không được thụt lề cách nhau một cách đồng nhất, Python sẽ báo lỗi `IndentationError`. Tương tự, nếu chúng ta sử dụng cả thụt lề và dấu ngoặc để định nghĩa một khối mã, cũng có thể gây lỗi.

Để sửa lỗi này, hãy kiểm tra và cân nhắc việc sắp xếp thụt lề sao cho đồng nhất trong toàn bộ mã nguồn của chúng ta. Sử dụng dấu tab hoặc dấu cách (nhưng không nên kết hợp cả hai) để thụt lề.

Ví dụ 1:

```
# Khai bao bien a va b
a = 5
  b = 6

# Thuc hien a + b, sau do luu vao
    c
c = a + b

# In gia tri c
print(c)
```

```
  File "<ipython-input-11-9facac1bc900>", line 3
    b = 6
    ^
IndentationError: unexpected indent
```

Trong ví dụ trên có vấn đề về thụt lề, dẫn đến lỗi `IndentationError`. Để sửa lỗi này chúng ta cần giữ cho thụt lề đồng nhất trong toàn bộ mã nguồn.

6. Module Not Found Error

Lỗi **ModuleNotFoundError** xuất hiện khi Python không thể tìm thấy một module (thư viện) mà chúng ta cố gắng import vào chương trình của mình. Module là một tập hợp các đoạn mã được đóng gói lại để sử dụng trong các chương trình khác nhau.

Có một số nguyên nhân có thể gây ra lỗi này:

- **Module không được cài đặt**: Chúng ta cần đảm bảo rằng module chúng ta muốn sử

dụng đã được cài đặt trên hệ thống. Chúng ta có thể sử dụng câu lệnh pip để cài đặt module: `pip install <module_name>`.

- **Đường dẫn không chính xác**: Nếu module chúng ta muốn sử dụng là một file riêng biệt thì chúng ta cần phải chỉ đúng đường dẫn của file đó.
- **Vấn đề với PYTHONPATH**: Đôi khi, lỗi này có thể xảy ra nếu biến môi trường `PYTHONPATH` không đúng.

Để khắc phục lỗi `ModuleNotFoundError`, chúng ta cần kiểm tra các điều kiện trên và kiểm tra lại module chúng ta muốn sử dụng có trong thư mục dự án hoặc đã được cài đặt chính xác hay chưa.

Ví dụ:

```python
import mymodule

print("aivietnam.ai")
```

```
---------------------------------------------------------------------------
ModuleNotFoundError                       Traceback (most recent call last)
<ipython-input-12-6901fb607fc9> in <cell line: 1>()
----> 1 import mymodule
      2
      3 print("aivietnam.ai")

ModuleNotFoundError: No module named 'mymodule'

---------------------------------------------------------------------------
NOTE: If your import is failing due to a missing package, you can
manually install dependencies using either !pip or !apt.

To view examples of installing some common dependencies, click the
"Open Examples" button below.
---------------------------------------------------------------------------
```

Trong chương trình trên, lỗi xảy ra do `mymodule` chưa được cài đặt hoặc không tìm thấy trong thư mục làm việc hiện tại.

7. Index Error

Lỗi **IndexError** xuất hiện khi chương trình cố gắng truy cập một chỉ mục trong một chuỗi, danh sách, hoặc tuple mà không tồn tại. Python sử dụng chỉ mục để xác định vị trí của các phần tử trong các cấu trúc dữ liệu này, và chỉ mục thường bắt đầu từ 0.

Ví dụ, nếu chúng ta có một danh sách với 5 phần tử và chúng ta cố gắng truy cập phần tử thứ 6 thông qua chỉ mục 5, Python sẽ báo lỗi `IndexError` vì chỉ mục 5 không tồn tại trong danh sách.

Để khắc phục lỗi `IndexError`, hãy kiểm tra các chỉ mục chúng ta đang sử dụng và đảm bảo rằng chúng không vượt quá kích thước của chuỗi, danh sách, hoặc tuple. Đặc biệt lưu ý rằng chỉ mục của cấu trúc dữ liệu thường bắt đầu từ 0 và kết thúc ở (số phần tử - 1).

Ví dụ 1:

```python
# Khai bao 1 list cac gia tri tu 1 den 5
l = [1, 2, 3, 4, 5]

print(l[0])
print(l[5])
```

```
1
---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
<ipython-input-13-2530f77aa9e8> in <cell line: 5>()
      3
      4 print(l[0])
----> 5 print(l[5])

IndexError: list index out of range
```

Trong ví dụ 1, chương trình đang thực hiện truy cập phần tử thứ 6 của list `l` bằng chỉ mục 5. Tuy nhiên, danh sách chỉ có 5 phần tử (chỉ mục từ 0 đến 4), nên Python sẽ báo lỗi `IndexError`.

Để sửa lỗi này, chúng ta sử dụng các chỉ mục từ 0 đến 4.

Ví dụ 2:

```
# Khai bao string
name = "aivietname.ai"

print(name[0])
print(name[50])
```

```
a
---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
<ipython-input-15-140310e7d47c> in <cell line: 5>()
      3
      4 print(name[0])
----> 5 print(name[50])

IndexError: string index out of range
```

Trong ví dụ 2, chương trình thực hiện truy cập phần tử thứ 51 của chuỗi `name` bằng chỉ mục 50. Nhưng độ dài của chuỗi chỉ là 15 ký tự, vì vậy chỉ mục 50 vượt quá kích thước của chuỗi, và Python sẽ báo lỗi `IndexError`. Để sửa lỗi này, chúng ta chỉ nên sử dụng các chỉ mục từ 0 đến 14.

8. Value Error

Lỗi **ValueError** xuất hiện khi một hàm hoặc toán tử nhận đối số có kiểu dữ liệu đúng, nhưng giá trị của đối số không nằm trong phạm vi được chấp nhận.

Ví dụ, nếu chúng ta sử dụng hàm `int()` để chuyển đổi một chuỗi thành một số nguyên và chuỗi đó không chứa một giá trị số hợp lệ, Python sẽ thông báo lỗi `ValueError`. Tương tự, một số hàm khác cũng có thể bị `ValueError` nếu đối số của chúng không phù hợp với yêu cầu của hàm.

Để khắc phục lỗi `ValueError`, chúng ta nên kiểm tra giá trị đang đưa vào hàm hoặc toán tử và đảm bảo rằng nó ở trong phạm vi chấp nhận được. Chúng ta cũng có thể kiểm tra kiểu dữ liệu của đối số để đảm bảo nó phù hợp với yêu cầu của hàm.

Ví dụ 1:

```
import math

number = -4
print(math.sqrt(number))
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-16-f25b4b744f6e> in <cell line: 4>()
      2
      3 number = -4
----> 4 print(math.sqrt(number))

ValueError: math domain error
```

Trong ví dụ 1, chương trình thực hiện tính căn bậc hai của một số âm (`number = -4`). Hàm `math.sqrt()` chỉ chấp nhận số không âm, vì vậy Python sẽ thông báo lỗi `ValueError`. Để khắc phục lỗi này, chúng ta phải sử dụng số lớn hơn hoặc bằng 0.

Ví dụ 2:

```
my_string = "Day la bai hoc cua AI
    VIETNAM"

my_string.index("hello")
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-23-016364efb00b> in <cell line: 3>()
      1 my_string = "Day la bai hoc cua AI VIETNAM"
      2
----> 3 my_string.index("hello")

ValueError: substring not found
```

Trong ví dụ trên, hàm `index()` sẽ trả về `ValueError` nếu giá trị chúng ta đang tìm kiếm (`"hello"`) không xuất hiện trong chuỗi `my_string`. Để tránh lỗi này, chúng ta cần kiểm tra xem giá trị đó có tồn tại trong chuỗi trước khi sử dụng hàm `index()`.

Ví dụ 3:

```
str1 = '5'
str2 = 'hello'

value1 = int(str1)
value2 = int(str2)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-18-aa88d3c207ee> in <cell line: 5>()
      3
      4 value1 = int(str1)
----> 5 value2 = int(str2)

ValueError: invalid literal for int() with base 10: 'hello'
```

Trong ví dụ 3, chương trình thực hiện chuyển đổi chuỗi `"hello"` thành một số nguyên bằng hàm `int()`, nhưng chuỗi này không chứa một giá trị số hợp lệ nên chương trình thông báo `ValueError`. Để sửa lỗi này, chúng ta cần đảm bảo rằng chúng ta chỉ chuyển đổi chuỗi chứa giá trị số.

9. Recursion Error

Lỗi **RecursionError** xuất hiện khi một chương trình Python sử dụng đệ quy (recursion) và quá trình đệ quy không kết thúc hoặc quá sâu, dẫn đến việc vượt quá giới hạn đệ quy (recursion limit). Mỗi lần hàm đệ quy được gọi, một frame mới được thêm vào stack của chương trình. Khi stack tràn, lỗi `RecursionError` sẽ xảy ra.

Ví dụ, nếu chúng ta viết một hàm đệ quy mà không có điều kiện dừng hoặc điều kiện dừng không đạt được, chương trình sẽ tiếp tục gọi đệ quy mãi mãi, cuối cùng dẫn đến lỗi `RecursionError`.

Để khắc phục lỗi `RecursionError`, chúng ta cần kiểm tra hàm đệ quy của mình và đảm bảo rằng có một điều kiện dừng hợp lý để chấm dứt quá trình đệ quy.

Ví dụ:

```
def a_function(n):
    return a_function(n)

a_function(5)
```

```
---------------------------------------------------------------------------
RecursionError                            Traceback (most recent call last)
<ipython-input-19-2f9feb1549f8> in <cell line: 4>()
      2     return a_function(n)
      3
----> 4 a_function(5)

 ⇕ 1 frames
... last 1 frames repeated, from the frame below ...

<ipython-input-19-2f9feb1549f8> in a_function(n)
      1 def a_function(n):
----> 2     return a_function(n)
      3
      4 a_function(5)

RecursionError: maximum recursion depth exceeded
```

Chương trình trên đang mô phỏng một hàm đệ quy vô hạn vì nó gọi chính nó mà không có điều kiện dừng. Điều này sẽ dẫn đến việc gọi đệ quy mãi mãi và cuối cùng thông báo lỗi `RecursionError`.

B. Lý thuyết Assert

assert là một từ khóa trong Python được sử dụng để kiểm tra điều kiện. Nếu điều kiện là True, mọi thứ diễn ra bình thường. Tuy nhiên, nếu điều kiện là False, chương trình sẽ đưa ra một lỗi `AssertionError` và dừng lại.

Chúng ta có cú pháp để sử dụng assert như sau:

```
assert expression, message
```

Trong đó:

- expression: Điều kiện cần được kiểm tra.
- message: Thông báo được hiển thị nếu điều kiện là False (không bắt buộc).

Ví dụ 1:

```
x = 5

assert x == 5, "The value of x is not 5"
print("The program continues to execute...")

>> Output: The program continues to execute...
```

Doạn mã trên kiểm tra xem giá trị của `x` có bằng `5` hay không bằng cách sử dụng `assert`, và sau đó in ra thông báo `"The program continues to execute..."` cho biết chương trình tiếp tục thực hiện.

Đối với trường hợp sau đây, `x` không phải là `5`, chương trình sẽ đưa ra một `AssertionError` và thông báo lỗi "`The value of x is not 5`" sẽ được hiển thị.

```
x = 2023
assert x == 5, "The value of x is
                not 5"
print("The program continues to
    execute...")
```

```
---------------------------------------------------------------------------
AssertionError                            Traceback (most recent call last)
<ipython-input-2-004472f7ab3d> in <cell line: 2>()
      1 x = 2023
----> 2 assert x == 5, "The value of x is not 5"
      3 print("The program continues to execute...")

AssertionError: The value of x is not 5
```

Ví dụ 2:

Chúng ta cũng có thể sử dụng `assert` để kiểm tra hàm. Trong ví dụ này chúng ta có hàm `divide(a, b)` thực hiện phép chia hai số, hàm này sử dụng `assert` để đảm bảo không thực hiện phép chia cho mẫu số bằng 0.

Nếu điều kiện b != 0 trong assert đúng thì phép chia được thực hiện và in ra mà hình kết quả.

```
def divide(a, b):
    assert b != 0, "The divisor cannot be 0"
    return a / b

result = divide(10, 2)
print("result:", result)

>> Output: result: 5.0
```

Nếu điều kiện sai, như trong trường hợp `divide(10, 0)` dưới đây, chương trình sẽ dừng lại với thông báo lỗi "`The divisor cannot be 0`".

```python
def divide(a, b):
    assert b != 0, "The divisor cannot be 0"
    return a / b

result = divide(10, 0)
print("result:", result)
```

```
AssertionError                            Traceback (most recent call last)
<ipython-input-4-b567508e6f8f> in <cell line: 5>()
      3     return a / b
      4
----> 5 result = divide(10, 0)
      6 print("result:", result)

<ipython-input-4-b567508e6f8f> in divide(a, b)
      1 def divide(a, b):
----> 2     assert b != 0, "The divisor cannot be 0"
      3     return a / b
      4
      5 result = divide(10, 0)

AssertionError: The divisor cannot be 0
```

Nhìn chung, **assert** là một công cụ mạnh mẽ trong Python giúp kiểm tra điều kiện trong quá trình lập trình và debug (thuật ngữ chỉ quá trình tìm vừa sửa lỗi trong code). Nó đặc biệt hữu ích để phát hiện sớm các vấn đề tiềm ẩn trong chương trình của bạn.

II. Trắc nghiệm

Câu hỏi 1: Đầu ra của chương trình sau đây là gì?

```
numbers = [1, 2, 3]
print(numbers[3])
```

a) 3
b) [1, 2, 3]
c) IndexError
d) None

Câu hỏi 2: Đầu ra của chương trình sau đây là gì?

```
def func():
    return 'Hello'
print(func(0))
```

a) H
b) TypeError
c) 'Hello'
d) SyntaxError

Câu hỏi 3: Đầu ra của chương trình sau đây là gì?

```
for i in range(5)
    print(i)
```

a) 0 1 2 3 4
b) SyntaxError
c) 0 1 2 3
d) None

Câu hỏi 4: Đầu ra của chương trình sau đây là gì?

```
x = 10
y = 0
print(x/y)
```

a) 0
b) Infinity
c) ZeroDivisionError
d) 10

Câu hỏi 5: Đầu ra của chương trình sau đây là gì?

```
name = 'Alice'
age = 30
print("Name: " + name + ", Age: " + age)
```

a) Name: Alice, Age: 30
b) TypeError
c) Name: , Age:
d) SyntaxError

Câu hỏi 6: Đầu ra của chương trình sau đây là gì?

```python
x = 5
if x == "5":
    print("Yes")
else:
    print("No")
```

a) No
b) Yes
c) TypeError
d) SyntaxError

Câu hỏi 7: Đầu ra của chương trình sau đây là gì?

```python
print(Hello)
```

a) Hello
b) NameError
c) 'Hello'
d) SyntaxError

Câu hỏi 8: Đầu ra của chương trình sau đây là gì?

```python
a = 5
b = "5"
if a + b:
    print("Math is fun")
```

a) Math is fun
b) 10
c) None
d) TypeError

Câu hỏi 9: Function sau nhận vào 1 list các số và trả về giá trị nhỏ nhất trong các số đó. Bạn hãy hoàn thành function và chọn phương án đúng.

```python
def get_min(data):
    min_value = data[0]

    ############ Your code here

    ############

    return min_value

data1 = [1, 5, 2, 91, -10]
assert get_min(data1) == -10

data2 = [2, 5, -5, 10, 11]
print(get_min(data2))
```

a) 3
b) -5
c) 2
d) 0

Câu hỏi 10: Function sau nhận vào 1 list các số và trả về giá trị lớn nhất trong các số đó. Bạn hãy hoàn thành function và chọn phương án đúng.

```python
def get_max(data):
    max_value = data[0]

    ############ Your code here

    ############

    return max_value

data1 = [1, 5, 2, 91, -10]
assert get_max(data1) == 91

data2 = [2, 5, -5, 10, 11]
print(get_max(data2))
```

a) 11
b) 5
c) 3
d) 10

Câu hỏi 11: Function sau nhận vào 1 list các số và 1 số cho trước, function trả về số lần xuất hiện của số đó trong list. Bạn hãy hoàn thành function và chọn phương án đúng.

```python
def count(data, value):
    count = 0

    ############ Your code here

    ############

    return count

data1 = [1, 5, 2, -10, 5, 10, 5]
assert count(data1, 5) == 3

data2 = [5, 4, 2, 10, 4, 8, 1, 3]
print(count(data2, 4))
```

a) 1
b) 0
c) 3
d) 2

Câu hỏi 12: Function sau nhận vào 1 list các số và trả về tổng của các số đó. Bạn hãy hoàn thành function và chọn phương án đúng.

```python
def get_sum(data):
    sum_value = 0

    ############ Your code here

    ############

    return sum_value

data1 = [1, 5, 2, 8, -1]
assert get_sum(data1) == 15

data2 = [5, 4, 2, 10, 4]
```

```
print(get_sum(data2))
```

a) 13
b) 25
c) 11
d) 22

Câu hỏi 13: Function sau nhận vào 1 list các số và trả về tổng của các số chẵn. Bạn hãy hoàn thành function và chọn phương án đúng.

```
def get_sum_even(data):
    sum_value = 0

    ############ Your code here

    ############

    return sum_value

data1 = [1, 5, 2, 4, -10, 5]
assert get_sum_even(data1) == -4

data2 = [5, 4, 2, 10, 3, 8]
print(get_sum_even(data2))
```

a) 22
b) 18
c) 24
d) 26

Câu hỏi 14: Function sau nhận vào 1 list các số và trả về giá trị trung bình của các số đó. Sử dụng hàm `round()` để làm tròn kết quả đến chữ số thập phân thứ 2. Bạn hãy hoàn thành function và chọn phương án đúng.

```
def get_mean(data):
    sum_value = 0

    ############ Your code here

    ############

    return mean

data1 = [1, 5, 2, 7, -3]
assert get_mean(data1) == 2.4

data2 = [2, 5, -5, 10, 11]
print(get_mean(data2))
```

a) 4.6
b) 3.2
c) 4.4
d) 3.8

Câu hỏi 15: Function sau nhận vào 1 list các số và 1 số cho trước, function kiểm tra số đó có xuất hiện trong list hay không và trả về giá trị True hoặc False. Bạn hãy hoàn thành function và chọn phương án đúng.

```python
def contain(data, value):
    is_contain = False

    ############ Your code here

    ############

    return is_contain

data1 = [1, 5, 2, 91, -10]
assert contain(data1, 5) == True

data2 = [2, 5, -5, 10, 11]
print(contain(data2, 3))
```

a) True
b) False