Gerar exceções

5 minutos

Agora que você tem uma boa compreensão dos rastreamentos e do tratamento de exceções, vamos examinar a geração de exceções.

Talvez você já conheça uma situação que pode causar uma condição de erro quando estiver escrevendo o código. Nessas situações, é útil gerar exceções que permitam a outros códigos perceber qual é o problema.

Gerar exceções também pode ajudar na tomada de decisões de outros códigos. Como vimos anteriormente, dependendo do erro, o código pode tomar decisões inteligentes para resolver, solucionar ou ignorar um problema.

Os astronautas limitam o uso de água a cerca de 11 litros por dia. Vamos criar uma função que, dependendo do número de astronautas, pode calcular a quantidade de água que restará após um dia ou mais:

Python

```
def water_left(astronauts, water_left, days_left):
    daily_usage = astronauts * 11
    total_usage = daily_usage * days_left
    total_water_left = water_left - total_usage
    return f"Total water left after {days_left} days is: {total_water_left}
liters"
```

Experimente com cinco astronautas, 100 litros de água restantes e dois dias pela frente:

Python

```
water_left(5, 100, 2)
```

Output

```
'Total water left after 2 days is: -10 liters'
```

Isso não é muito útil, pois um déficit de litros deve ser considerado um erro. Então, o sistema de navegação pode alertar os astronautas de que não haverá água suficiente para todos em

dois dias. Se você for um engenheiro que está programando o sistema de navegação, poderá gerar uma exceção na função `water_left()` para alertar a condição de erro:

Python

```
def water_left(astronauts, water_left, days_left):
    daily_usage = astronauts * 11
    total_usage = daily_usage * days_left
    total_water_left = water_left - total_usage
    if total_water_left < 0:
        raise RuntimeError(f"There is not enough water for {astronauts} astronauts
after {days_left} days!")
    return f"Total water left after {days_left} days is: {total_water_left}
liters"
```

Agora, execute novamente:

Python

```
water_left(5, 100, 2)
```

Output

```
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "<stdin>", line 6, in water_left
RuntimeError: There is not enough water for 5 astronauts after 2 days!
```

No sistema de navegação, o código para sinalizar o alerta agora pode usar `RuntimeError` para alertar:

Python

```
try:
    water_left(5, 100, 2)
except RuntimeError as err:
    alert_navigation_system(err)
```

A função `water_left()` também pode ser atualizada para evitar a passagem de tipos sem suporte. Tente passar argumentos que não são inteiros para verificar a saída de erro:

Python

```
water_left("3", "200", None)
```

Output

```
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "<stdin>", line 3, in water_left
TypeError: can't multiply sequence by non-int of type 'NoneType'
```

O erro de `TypeError` não é muito amigável no contexto do que a função espera. Atualize a função para que ela use `TypeError`, mas com uma mensagem melhor:

Python

```
def water_left(astronauts, water_left, days_left):
    for argument in [astronauts, water_left, days_left]:
        try:
            # If argument is an int, the following operation will work
            argument / 10
        except TypeError:
            # TypeError will be raised only if it isn't the right type
            # Raise the same exception but with a better error message
            raise TypeError(f"All arguments must be of type int, but received:
'{argument}'")
    daily_usage = astronauts * 11
    total_usage = daily_usage * days_left
    total_water_left = water_left - total_usage
    if total_water_left < 0:
        raise RuntimeError(f"There is not enough water for {astronauts} astronauts
after {days_left} days!")
    return f"Total water left after {days_left} days is: {total_water_left}
liters"
```

Agora, tente novamente para obter um erro melhor:

Python

```
water_left("3", "200", None)
```

Output

```
Traceback (most recent call last):
  File "<stdin>", line 5, in water_left
TypeError: unsupported operand type(s) for /: 'str' and 'int'

During handling of the preceding exception, another exception occurred:

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

```
  File "<stdin>", line 9, in water_left
TypeError: All arguments must be of type int, but received: '3'
```

Unidade seguinte: Exercício – Trabalhar com exceções

Continuar >