exceptions: знайомство і призначення

Виключення (exceptions) - це об’єкти в Python які призначені для реалізації поведінки керування потоком виконання коду і дозволяють “перескакувати” значні блоки коду. Вони мають специфічні атрибути для реалізації цієї поведінки і використовуються в певних ситуаціях як автоматично (“під капотом” - наприклад - при виникненні помилок під час виконання коду) так і можуть бути ініційовані розробником за допомогою спеціальних операторів.

```
>>> def division(a, b) -> float:
...     result = a / b
...     print(f"{a} divided by {b} is
{result}")
...     return result
...
>>> division(2, 5)
2 divided by 5 is 0.4
0.4
>>> division(2, 0)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "<stdin>", line 2, in division
ZeroDivisionError: division by zero
>>>
```

створмо функцію, яка реалізує операцію поділу двох чисел

під час “звичайних викликів” все працює відповідно коду який ми описали в тілі функції

якщо у якості дільника (другий аргумент) буде передано 0, в тілі функції буде поділ на 0. Це невизначена операція у Python - автоматично буде згенеровано виключення, яке призведе до зупинки виконання коду, і створення відповідного повідомлення в консолі.

Приклад показав нам автоматичну генерацію виключення у певній ситуації. Як правило, ми отримуємо в повідомленні тип виключення (**ZeroDivisionError** - в нашому випадку), додаткове повідомлення (“**divsion by zero**” - у нас), і вказівник на місце в коді яке ініціювало генерацію виключення (його називають **Traceback**).
Ви, напевно, зустрічали різні виключення (**ValueError**, **TypeError**, **KeyError**, **IndexError**, …). Існує доволі велика кількість вбудованих типів виключень у Python які ініціюються у відповідних ситуаціях.

exceptions: вбудовані класи виключень

Python має доволі велику кількість вбудованих винятків, які призначені для використання в різних ситуаціях.

```
>>> ZeroDivisionError.mro()
[<class 'ZeroDivisionError'>, <class
'ArithmeticError'>, <class 'Exception'>, <class
'BaseException'>, <class 'object'>]
```

Виключення (exceptions) - це об’єкти які мають бути екземплярами класу, який походить від вбудованого класу **BaseException**. Ми можемо це побачити використовуючи вже знайомий нам функціонал (напри клад метод **mro**()) для якогось вже знайомого нам виключення.

Для кожної конкретної ситуації може бути створений свій екземпляр класу відповідного винятку і він буде нести в собі специфічну інформацію саме для цієї ситуації. Зверніть увагу що при передачі, наприклад, функції len() аргументів для яких вона не може бути визначена, ми маємо різні повідомлення про помилку: тому що були створені різні екземпляри винятку **TypeError** який було згенеровано в цій ситуації.

```
>>> len("test")
4
>>> len(2)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: object of type 'int' has no
len()
>>> len(False)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: object of type 'bool' has no
len()
>>>
```

exceptions: коли і для чого використовуються

Виключення невірно сприймати виключно як інструмент для роботи з помилками, їх використання набагато більш широке: це специфічний інструмент який дає можливість перескакувати через довільно велику кількість коду. Це використовується у наступних ситуаціях:

- обробка помилок
  - інтерпретатор Python генерує виключення щоразу коли зустрічає помилки в програмах під час виконання.
- повідомлення про якісь події
  - виключення можуть використовуватись для повідомлень про якісь певні події. Вам вже знайома одна з таких ситуацій - генерація виключення StopIteration при вичерпанні ітератора.
- обробка особливих випадків
  - інколи якась особлива ситуація може виникати так не часто, що писати код для її обробки в основному коді - це лише вносити в код додаткову заплутаність. Можливий вихід - при виникненні такої ситуації генерація виключення і обробка цього виключення - а, відповідно, і цієї особливої ситуації - в окремому обробнику.
- обов’язкові дії при завершенні
  - інструменти роботи з виключеннями надають можливість створювати блоки коду які будуть обов’язково виконані інтерпретатором під час завершення роботи програми в будь-якому випадку, незалежно від того штатне завершення коду чи по виникненню якогось неочікуваного/очікуваного виключення.