РОЗДІЛ 6. ВИНЯТКИ

6.1. Загальні поняття

Винятки – це сповіщення інтерпретатора, порушувані в разі виникнення помилки в програмному коді або при настанні якої-небудь події. Якщо в коді не передбачено оброблення винятків, то програма переривається і виводиться повідомлення про помилку.

Нагадаємо, що існує три типи помилок в програмі:

Синтаксичні – це помилки в імені оператора або функції, невідповідність закриваючих та відкриваючих лапок і т.д. Тобто помилки в синтаксисі мови. Як правило, інтерпретатор попередить про наявність помилки, а програма не виконуватиметься зовсім. Приклад синтаксичної помилки:

```
print("Невідповідність відкритих та закритих лапок!)
```

Результатом запуску даного коду буде:

```
SyntaxError: EOL while scanning string literal
```

Семантичні – це помилки в логіці роботи програми, які можна виявити тільки за результатами роботи скрипта. Як правило, інтерпретатор не попереджає про наявність помилки. А програма буде виконуватися, оскільки не містить синтаксичних помилок. Такі помилки досить важко виявити і виправити.

Помилки часу виконання – це помилки, які виникають під час роботи скрипта. Причиною є події, які не передбачені програмістом. Класичним прикладом служить ділення на нуль:

```
def test (x, y):
    return x/y

print(test(4, 2))
```

Результатом запуску даного коду буде:

```
Traceback (most recent call last):
  File "G:\КПИ\2017-2018\Иностранцы\lab2.py", line 4, in
<module>
    print(test(4, 0))
```

```
  File "G:\lab.py", line 2, in test
    return x/y
ZeroDivisionError: division by zero
```

В мові Python винятки порушуються не тільки при помилці, але і як повідомлення про настання будь-яких подій. Наприклад, метод *index()* збуджує виняток *ValueError*, якщо шуканий фрагмент не входить в рядок:

```
>>> "String". index ("S")
0

>>> "String". index ("s")
Traceback (most recent call last):
  File "<pyshell#13>", line 1, in <module>
    "String". index ("s")
ValueError: substring not found
```

6.2. Оброблення винятків

Для оброблення винятків призначена інструкція ***try*** [5, 8, 9, 13]. Формат інструкції:

try:
<БЛОК, В ЯКОМУ ПЕРЕХОПЛЮЮТЬСЯ ВИНЯТКИ>
except <ВИНЯТОК_1> as <ОБ'ЄКТ ВИНЯТКУ>:
<БЛОК, ЩО ВИКОНУЄТЬСЯ ПРИ ЗБУДЖЕННІ ВИНЯТКУ>
…
except <ВИНЯТОК_N> as <ОБ'ЄКТ ВИНЯТКУ>:
<БЛОК, ЩО ВИКОНУЄТЬСЯ ПРИ ЗБУДЖЕННІ ВИНЯТКУ>
e1se:
<БЛОК, ЩО ВИКОНУЄТЬСЯ, ЯКЩО ВИНЯТКУ НЕ ВИНИКЛО>
finally:
< БЛОК, ЩО ВИКОНУЄТЬСЯ В БУДЬ-ЯКОМУ ВИПАДКУ>

Інструкції, в яких перехоплюються винятки, повинні бути розташовані всередині блоку *try*. У блоці *except* в параметрі <Виняток_1> вказується клас оброблюваного винятку.

Наприклад, щоб обробити виняток, що виникає при діленні на нуль:

```
try:                              # Перехоплюється виняток
    x=1/0                             # Помилка: ділення на 0
except  ZeroDivisionError:        # Вказуємо клас винятку
    print ("Обробили ділення на 0")
    x=0
    print(x)
```

Результатом запуску даного коду буде:

```
Обробили ділення на 0
0
```

Якщо в блоці *try* згенеровано виняток, то управління передається блоку *except*. У разі, якщо виключення не відповідає зазначеному класу, управління передається наступному блоку *except*. Якщо жоден блок *except* не відповідає винятку, то виняток "спливає" до обробника більш високого рівня. Якщо виняток ніде не обробляється в програмі, то управління передається обробнику за замовчуванням, який зупиняє виконання програми і виводить стандартну інформацію про помилку. Таким чином, в обробнику може бути кілька блоків *except* з різними класами винятків. Крім того, один обробник можна вкласти в інший.

```
try:                              #  Обробляється виняток
    try:                          #  Вкладений обробник
       x=1/0                      #  Помилка: ділення на 0
    except NameError:
       print ("Невизначений ідентифікатор")
    except IndexError:
       print ("неіснуючий індекс")
    print ("Вираз після вкладеного обробника")
except  ZeroDivisionError:
    print ("Обробка ділення  на 0")
    x=0
```

```
    print(x)
```

Результатом запуску даного коду буде:

```
Оброблення ділення  на  0
0
```

У вкладеному обробнику не вказано виняток *ZeroDivisionError*, тому виняток "спливає" до обробника більш високого рівня. Після оброблення винятку управління передається інструкції, розташований відразу після обробника.

В інструкції *except* можна вказати відразу кілька винятків, перерахувавши їх через кому всередині круглих дужок.

```
try:
    x = 1/0
except (NameError, IndexError, ZeroDivisionError):

# Оброблення відразу декількох винятків

    x  = 0
print (x)
```

Якщо в інструкції ***except*** не вказано клас винятку, то такий блок перехоплює всі винятки.

```
try:
    x = 1/0
except:                                  # Оброблення всіх винятків
    x = 0
    print (x)
```

На практиці слід уникати порожніх інструкцій *except*, оскільки можна перехопити виняток, яке є лише сигналом системі, а не помилкою.

Якщо в обробнику присутній блок *e1se*, то інструкції всередині цього блоку будуть виконані тільки при відсутності помилок. При необхідності виконати будь-які завершальні дії незалежно від того, було згенеровано виняток чи ні, слід скористатися блоком *fina11y*.

```
try:
   x  = 10/2                          #  Немає помилки
   #x  = 10/0                         #  Помилка: ділення на 0
except  ZeroDivisionError:
   print ("Ділення на 0")
else:
   print ("Блок e1se")
finally:
   print ("Блок finally")
```

Результат виконання при відсутності винятку:

```
Блок  e1se
Блок  finally
```

Послідовність виконання блоків при наявності винятку буде інший:

```
Ділення на 0
Блок  f1nally
```

При наявності винятку і відсутності блоку *except* інструкції всередині блоку *finally* будуть виконані, але виняток не буде оброблено. Він продовжить "спливання" до обробника більш високого рівня.

```
try:
  x  = 10/0
finally:
  print ("Блок  fina11y")
```

Якщо користувацький обробник відсутній, то управління передається обробнику за умовчанням, який перериває виконання програми і виводить повідомлення про помилку.

```
Блок  fina11y
Traceback (most recent call last):
  File "G:\lab.py", line 2, in <module>
    x  = 10/0
ZeroDivisionError: division by zero
```

6.3. Класи вбудованих винятків

Всі вбудовані виключення в мові Python представлені у вигляді класів. Ієрархія вбудованих класів винятків:

```
BaseException
 GeneratorExit
 Keyboardinterrupt
 SystemExit
 Exception
    Stopiteration
    Warning
       BytesWarning,  ResourceWarning,
       DeprecationWarning,  FutureWarning,  ImportWarning,
       PendingDeprecationWarning,  RuntimeWarning,
SyntaxWarning,
       UnicodeWarning,  UserWarning
    Ar i trЛlet icError
       FloatingPointError,  OverflowError,  ZeroDivisionError
    AssertionError
    AttributeError
    BufferError
    EnvironmentError
       IOError
       OSError
          WindowsError
    EOFError
    ImportError
    LookupError
       IndexError,  KeyError
    MemoryError
    NameError
       UnboundLocalError
    ReferenceError
    RuntimeError
       NotimplementedError
    SyntaxError
       IndentationError
```

```
            TabError
      SystemError
      TypeError
      ValueError
         UnicodeError
            UnicodeDecodeError,  UnicodeEncodeError
            UnicodeTranslateError
```

Основна перевага використання класів для оброблення винятків полягає в можливості вказівки базового класу для перехоплення всіх винятків відповідних класів-нащадків. Наприклад, для перехоплення ділення на нуль було використано клас *ZeroDivisionError*. Якщо замість цього класу вказати базовий клас *ArithrneticError*, то перехоплюватимуться винятки класів *FloatingPointError*, *OverflowError* і *ZeroDivisionError*.

```
try:
   x  = 1/0
except  ArithmeticError:                    # Вказано базовий клас
   print ("Обробили ділення  на 0")
```

Запитання для самоконтролю

1. Поняття помилки.
2. Що таке виняткові ситуації і яким чином здійснюється їх оброблення у Python?
3. Блоки try – except.
4. Атрибути винятків, ініціювання винятків.
5. Для чого використовується гілка finally в інструкції try?
6. Чи можна гілку finally поєднувати з гілками except?
7. Які два класи виняткових ситуацій наявні в Python?
8. Який із класів виняткових ситуацій рекомендується використовувати у програмах?