Технически Университет – София,
Факултет „Компютърни системи и технологии“

Упражнение 7 – Тестване на програмен код. Грешки. Прихващане и обработка на изключителни ситуации.

Тестване и отстраняване на грешки

Тъй като в Python липсва етап компилация, цикълът редактиране (edit) тестване (test) - отстраняване на грешки (debug) е бърз. Когато интерпретаторът открие грешка, той поражда изключение. Когато програмата не засече изключението, интерпретаторът отпечатва проследяване на стека. Отстраняването на грешки на ниво източник позволява проверка на локални и глобални променливи, оценка на произволни изрази, задаване на точки на прекъсване (breakpoints). Програмата, занимаваща се с отстраняването на грешки в Python(debugger) е написана на самия Python език.

Грешки и изключителни ситуации

Синтактичните грешки се появяват, когато някое синтактично правило на езика е нарушено. За тях сигнализира интерпретаторът. Например по-надолу е показано съобщение за синтактична грешка, която е открита при функцията print(), тъй като двете точки пред нея липсват. Името на файла и номерът на реда са отпечатани също в съобщението за грешка, за да се ориентараме полесно за отстраняване на грешките.

```
while True print('Hello world')
  File "<stdin>", line 1
    while True print('Hello world')
                     ^
SyntaxError: invalid syntax

while True: print('Hello world')
```

При изключителните ситуации изразите са синтактично правилни, но възниква някаква грешка по време на изпълнение на кода. Те могат да се обработват от нашата програма или не. Когато не се обработват от нашата програма, се извеждат съобщения за грешка. Например, възможно е да се получи някое от следните съобщения

```
>>> 10 * (1/0)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ZeroDivisionError: division by zero
```

"<stdin>" – това означава, че кодът **не е в .py файл**, а е написан **директно в конзолата**

```
>>> 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
```

Последният ред на съобщението за грешка посочва какво се е случило. Има различни типове изключителни ситуации. В горните примери те са съответно: ZeroDivisionError, NameError и TypeError. Това са примери за така наречените вградени изключения. Техните имена са вградени идентификатори, а не ключови думи. Останалата част от реда, на който се появяват, показва подробности за типа изключителна ситуация и причината за нея. Началната част от съобщението за изключение пък показва контекста на изключението под формата на проследяване на стека (stack traceback). Най-общо се визуализират редове от кода, който се изпълнява.

В Python всички изключения са инстанции на клас, който наследява класа BaseException. Този клас се наследява от вградените изключения. Потребителски генерираните изключения обаче наследяват класа Exception.

Прихващате и обработка на изключителни ситуации

Блокът try позволява да тестваме код за грешки. **Блокът except** позволява да се обработват грешките. **Блокът finally** позволява да изпълним код, независимо от резултата на блоковете try и except.

Котато се появи грешка или предизвикано от нас изключение, Python обикновено спира и генерира съобщение за грешка.

Изключенията се обработнват с помощта на израза try.

Например в следващия код се генерира изключение, защото x не е дефинирано.

```
try:
    print(x)
except:
   print("An exception occurred")
```

Ако print(x) се изпълни извън блока try, без да е дефинирано, това ще предизвика грешка в изпълнението на програмата:

```
print(x)
 Traceback (most recent call last):
  File "main.py", line 1, in <module>
   print(x)
NameError: name 'x' is not defined
```

За един try блок може да се дефинират множество блокове except, които да обработват различни видове грешки. Например може да се изведе едно съобщение за грешка от тип NameError и друго за дргите получени грешки чрез следния код:

```
try:
    print(x)
except NameError:
    print("Variable x is not defined")
except:
   print("Something else went wrong")
```

Може да се използва ключовата дума else, за да се дефинира блок от код, който да се изпълни, ако не се появят грешки. Например следният блок от код:

```
try:
  print("Hello")
except:
  print("Something went wrong")
else:
  print("Nothing went wrong")
```

не генерира изключение и ще изведе съобщението ("Nothing went wrong"). Клаузата else трябва да бъде след всички except клаузи. Полезна е, когато трябва да се изпълни код, ако не настъпи изключение, както е показано в следния пример:

```
import sys

for arg in sys.argv[1:]:
    try:
        f = open(arg, 'r')
    except OSError:
        print('cannot open', arg)
    else:
        print(arg, 'has', len(f.readlines()), 'lines')
        f.close()
```

Този код взима всички аргументи от командния ред (имената на файлове), които са подадени след името на програмата – това е **sys.argv[1:].** За всеки такъв аргумент **arg** той се опитва **(try**) да отвори файла с име **arg** в режим за четене. Ако файлът не може да се отвори (например не съществува), влиза в **except OSError** и отпечатва съобщение cannot open .... Ако няма грешка, влиза **в else,** прочита всички редове на файла, отпечатва колко реда има, и накрая затваря файла с **f.close()**
Използването на **else** клауза е по-добър вариант от добавянето на допълнителен код в **try** секцията, защото така се избягва прихващане на изключение, което не е било придизвикано от кода, който се защитава срещу изключения.

Блокът **finally** ще се изпълни, независимо дали блокът **try** генерира или не изключение:

```
try:
   print(x)
except:
   print("Something went wrong")
finally:
   print("The 'try except' is finished")
```

Този блок може да се използва за затваряне на обектите и освобождаване на ресурсите. Например, ако програмата се опита да пише във файл, в който не е допустимо писане, следният код ще позволи да продължим, без да оставяме файла отворен.

```
try:
  f = open("demofile.txt")
  f.write("Lorum Ipsum")
except:
  print("Something went wrong when writing to the file")
finally:
  f.close()
```

Този код се опитва **(try)** да отвори файла **demofile.txt** и да запише в него текста **"Lorum Ipsum".** Ако при отварянето или писането във файла възникне грешка, се изпълнява блокът **except** и се отпечатва съобщението "**Something went wrong when writing to the file**". Блокът finally се изпълнява винаги – независимо дали е имало грешка или не. В него файлът се затваря с f.close(), за да се освободи ресурсът.

Следващият пример изисква от потребителя да въведе цяло число като вход:

```
while True:
   try:
     x = int(input("Please enter a number: "))
    break
  except ValueError:
     print("Oops!  That was no valid number.  Try
again...")
```

Цикълът **while True** кара програмата да пита потребителя за число отново и отново. В блока **try** програмата се опитва да преобразува въведеното с

int(...) в цяло число. Ако въведеното не е валидно число, възниква грешка **ValueError** и се изпълнява **except,** който показва съобщение „Oops! That was no valid number. Try again...“. Когато потребителят въведе коректно число, няма грешка **и break** прекъсва цикъла.

При изпълнението на кода в try блока, ако не настъпи изключителна ситуация, клаузата except се пропуска и изпълнението на израза try завършва. Ако възникне изключителна ситуация по време на изпълнение на кода в секцията try, останалата част от кода в секцията се изпуска и ако изключението съвпада с посоченото в клаузата except, се изпълнява кодът от тази клауза. След това изпълнението на програмата продължава от мястото след израза try. Ако настъпилата изключителна ситуация не съвпадне с посочената в клаузата except, тя се предава за обработка към други изрази try. Ако не се открие нито един try израз за нея, се получава неприхванато изключение (unhandled exception), изпълнението спира и се появява съответно съобщение.

Един израз try може да има няколко except клаузи, съответстващи на различни изключителни ситуации. Изпълнява се обаче най-много една клауза except. Една клауза except може да описва няколко изключения, разделени със запетая и оградени в скоби. Например:

```
... except (RuntimeError, TypeError, NameError):
...     pass
```

Когато в try има функция и при нейното извикване възникне изключителна ситуация, тази изключителна ситуация също се обработва от try израза. Например:

```
def this_fails():
     x = 1/0
try:
   this_fails()
except ZeroDivisionError as err:
print('Handling run-time error:', err)

#Handling run-time error: division by zero
```

Функцията **this_fails()** се опитва да изчисли 1/0, което винаги води до грешка деление на нула **(ZeroDivisionError).** Тази функция се извиква в блока try, където Python „пробва“ да изпълни кода. Когато възникне

грешката **ZeroDivisionError**, управлението преминава в блока **except ZeroDivisionError as err.** Там се отпечатва съобщението **Handling run-time error**: заедно с текста на грешката (err), например **division by zero.**

Възможно е изключителни ситуации да се генерират от програмиста по зададени от него критерии, т. е. при настъпване на определени условия. Това става с ключовата дума **raise.** След това се посочва какъв тип грешка да се генерира – Exception или TypeError. Например в кода по-долу се генерира грешка, ако стойността на променливата x стане по-малка от 0.

```
x = -1
if x < 0:
     raise Exception("Sorry, no numbers below zero")
```

В следващия пример се генерира грешка TypeError, ако типът на x не е int.

```
x = "hello"
if not type(x) is int:
     raise TypeError("Only integers are allowed")
```

Следващият пример показва какво се връща от интерпретатора при генериране на изключителната ситуация HiThere:

С raise казваме на Python: „сега умишлено
вдигни тази грешка“.

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

Единственият аргумент при извикване на raise показва кое изключение да се генерира. Той може да бъде както инстанция, така и клас на изключение (клас, която наследява базовия клас Exception). Ако се извика изключение клас, то имплицитно се инстанциира (създава се обект от този клас), като се извиква конструктор без аргументи. Така двете извиквания

raise ValueError

и

raise ValueError()

за класа изключение ValueError са еквивалентни.

Когато използваме **raise**, казваме на Python да генерира грешка от даден тип.

Тази „грешка“ трябва да е **обект** (инстанция) на някакъв **клас за изключение** – например ValueError.

Има два еднакви начина да го направим:

raise ValueError- Тук му даваме **само класа**. Python сам си казва: ще си създам обект от този клас: ValueError().“

raise ValueError() - Тук ние **сами** създаваме обекта ValueError() и го подаваме на raise.

И в двата случая резултатът е **същият** – хвърля се една и съща грешка ValueError без допълнително съобщение. Затова казваме, че:

raise ValueError и raise ValueError()са **еквивалентни** (един и същи ефект).

Новите класове изключения наследяват класа Exception директно или индиректно. При създаване на модули, които могат да генерират няколко различни грешки, често се дефинира базов клас изключение в този модул, след което се дефинират подкласове за различните видове изключения. Например:

```
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

class TransitionError(Error):
    """Raised when an operation attempts a state transition that's not allowed.

    Attributes:
        previous -- state at beginning of transition
        next -- attempted new state
        message -- explanation of why the specific transition is not allowed
    """
    def __init__(self, previous, next, message):
        self.previous = previous
        self.next = next
        self.message = message
```

Най-често имената на дефинираните изключения включват в края си “Error”.

Задачи :

1. Да се създаде клас BankAccount, който представя банкова сметка.
Класът трябва да има:
Полета (атрибути):

- **owner** – име на собственика
- **balance** – текущ баланс (число, по начало 0)

Методи:
deposit(amount) – увеличава баланса със сумата amount, ако amount е **положително число**, добавя го към баланса в противен случай да отпечата съобщение, че сумата е невалидна.

withdraw(amount) – намалява баланса със сумата amount, ако има **достатъчно пари по сметката**, да изтегли сумата, ако няма достатъчно средства, да отпечата съобщение, че балансът е недостатъчен.
print_info() – отпечатва името на собственика и текущия баланс.
Да се създаде **обект** от класа BankAccount, да се извикат методите deposit() и withdraw(), и накрая да се извика print_info(), за да се види крайният баланс.

2. Напишете код на метод, който приема като параметър име на текстов файл, прочита съдържанието на файла и го връща като стринг. Какво е правилно да направи методът с възникващите изключения?

3. Напишете програма, която прочита от конзолата цяло положително число и отпечатва на конзолата корен квадратен от това число. Ако числото е отрицателно или невалидно, да се изпише “Invalid Number”. Във всички случаи накрая да се изпише “Good Bye”.