**4. Винятки та їх обробка**

Виняток (exception) – це аварійний стан, який відбувається в кодовій послідовності під час виконання програми. Прикладом є — ділення на нуль, помилки читання з файлу, вичерпання доступної пам’яті тощо. Іншими словами – це помилки, які можуть виникнути при виконанні програми. В ряді мов програмування необхідно заздалегідь передбачити можливість тієї чи іншої помилки і визначити шлях її обробки. В Python для цього передбачений спеціальний механізм винятків.

Про винятки в Python можна говорити як про тип даних, що містить інформацію про помилки. На рівні з поняттям виняток можна зустріти поняття виняткова ситуація, тобто випадок, коли виник виняток, проте досить часто ці два поняття використовуються як синоніми.

## Винятки

Розглянемо як приклад, виняток ділення на нуль. Якщо спробувати виконати операцію, 1/0 то виникне помилка, оскільки на 0 ділити не можна.

Інтерпретатор відреагує на цю помилку генерацією винятку (припиненням подальшого виконання програми) та виведе відповідне повідомлення.

>>> 1/0

Traceback (most recent call last):

File "<pyshell#0>", line 1, in <module>

1/0

ZeroDivisionError: division by zero

Розберемо це повідомлення докладніше:

* Traceback (most recent call last) – повідомлення про те, що інтерпретатор «зловив» виняток;
* File "<pyshell#0>", line 1, in <module> – звідки було запущено код програми, в якому виник виняток (це може бути ім’я файлу, вказівка на інтерактивний режим в консолі – stdin, вказівка на інтерактивний режим оболонки IDLE - pyshell#0) і номер рядка, в якому це сталося;
* 1/0 – вираз, в якому стався виняток;
* ZeroDivisionError: division by zero – тип винятку (назва винятку) і його короткий опис.

Наведемо приклади деяких інших винятків:

***Операція застосована до об'єкту невідповідного типу:***

>>> 2 + '1'

Traceback (most recent call last):

File "<pyshell#0>", line 1, in <module>

2 + '1'

TypeError: unsupported operand type(s) for +: 'int' and 'str'

***Функція отримує аргумент правильного типу, але некоректного значення:***

>>> int('qwerty')

Traceback (most recent call last): File "<pyshell#0>", line 1, in <module> int('qwerty')

ValueError: invalid literal for int() with base 10:

'qwerty'

***Синтаксична помилка:***

>>> t:=3

File "<pyshell#0>", line 1, in <module> t:=3

^

SyntaxError: invalid syntax

У цих прикладах генеруються винятки: TypeError, ValueError, SyntaxError. Ієрархію вбудованих в Python винятків подано в додатку 4.

## Обробка винятків

Про винятки можна ще сказати, що це помилки, які можна обробити. Обробка винятків або обробка виняткових ситуацій – механізм мови програмування, призначений для опису реакції програми на помилки часу виконання та інші можливі проблеми, які можуть виникати при виконанні програми і призвести до неможливості (безглуздості) подальшого відпрацювання програмою її базового алгоритму.

Обробка винятків може полягати у гарантованому виконанні певного коду або в корекції стану програми, що викликав виняток. Особливо гостро питання надійності постають у програмах, пов’язаних з обробкою математичних об’єктів, таких, як наприклад функції, оскільки існує проблема області визначення, наявність розривів тощо, що призводить до помилок у обчисленнях з плаваючою точкою. Якісна програма повинна обробити такі помилки, присвоївши відповідній змінній певне значення або, якщо це необхідно, повідомивши про виникнення помилки користувача.

Знаючи, в яких місцях і за яких обставин можуть виникнути винятки, ми можемо передбачити їх обробку. Для обробки винятків використовується конструкція try-except, яка має декілька форм. В конструкції try-except можуть бути присутні оператори: try, except, else, finally, raise. Проте про них по порядку.

Найпростіший варіант конструкції try - except для обробки винятку має вигляд: try:

#Код блоку try (код, в якому може виникнути виняток)

except Назва\_винятку:

#Код блоку except (код, що виконується при вказаному винятку)

Вказана конструкція try - except виконується таким чином:

* Спочатку виконується код блоку try (вираз чи вирази між ключовими словами try і except).
* Якщо під час виконання коду блоку try не відбулося ніякого винятку, код блоку except пропускається і виконання конструкції try - except закінчено.
* Якщо під час виконання коду блоку try виникає виняток, виконання коду блоку try припиняється і управлїння переходить до обробника except.
* Якщо тип винятку, що виник, відповідає назві винятку, вказаного після ключового слова except, виконується код блоку except. Після завершення виконання коду блоку except виконується код, що міститься після конструкції try - except.
* Якщо виник виняток, тип якого не збігається з назвою типу винятку, вказаного після ключового слова except, виняток передається на зовнішню конструкцію try - except, якщо зовнішня конструкція відсутня чи обробник не знайдений і там, то виняток стає необробленим і виконання програми зупиняється з системним повідомленням про помилку, як показано вище.

Отже, в блоці try розміщується код, в якому може виникнути виняток, а в блоці except розміщується код для обробки відповідного винятку.

Незважаючи на те, що в приведеному варіанті конструкції try – except вказано лише єдину назву винятку, насправді буде оброблений як сам виняток, назву якого вказано, так і його нащадки. Наприклад, при перехопленні винятку ArithmeticError також будуть перехоплюватися і винятки FloatingPointError, OverflowError, ZeroDivisionError.

Якщо необхідно однаково обробити винятки, ієрархічно не пов’язані між собою, у рядку except можна перерахувати назви кількох таких винятків, взявши їх в дужки. Наприклад: try:

#Код блоку try (код, в якому можуть виникнути винятки)

except (RuntimeError, TypeError, NameError):

#Код блоку except (код, що виконується при вказаних винятках)

Також можливе застосування ключового слова except без указання назв винятків. Такий обробник буде перехоплювати всі винятки (в тому числі і системні: переривання з клавіатури, системний вихід і т.д.), і тому в такій формі except практично не використовується. В тих випадках, коли є необхідність в обробці всіх вбудованих несистемних винятків, може бути використаний запис except Exception.

В тих випадках, коли необхідно опрацювати по-різному різні винятки, що можуть виникнути в певному операторі (наборів операторів), в конструкції try - except можна розмістити декілька відповідних блоків except. Тоді при виникненні винятку будуть переглянуті блоки except по черзі, зверху до низу, в пошуках обробника відповідного винятку (тільки перший обробник, що підходить, буде виконаний).

Виходячи з сказаного конструкцію try – except можна представити в наступному вигляді: try:

#Код блоку try except Назва\_винятку1:

#Код блоку, що виконується при вказаних винятках except (Назва\_винятку2, Назва\_винятку3, …):

#Код блоку, що виконується при вказаних винятках . . .

except Exception:

#Код блоку, що виконується за будь-якого не

системного винятку, якого не було оброблено

Але й на цьому етапі вказана конструкція try – except не є повною. Конструкція try – except може мати ще два блоки: finally та else. Код блоку finally виконується в будь-якому випадку, незалежно від того, чи виник виняток в блоці try, чи ні. Код блоку else виконується в тому випадку, якщо винятку в блоці try не було. Блок else досить добре описує частину дерева розв’язку: «Якщо цього виконати не можна, то (інакше) виконати це».

Якщо блок else присутній, то він має йти після всіх блоків except, але до блоку finally . try:

#Код блоку try except Назва\_винятку1:

#Код блоку, що виконується при вказаних винятках except (Назва\_винятку2, Назва\_винятку3, …):

#Код блоку, що виконується при вказаних винятках . . . except Exception:

#Код блоку, що виконується за будь-якого не системного винятку, якого не було оброблено

else:

#Код блоку, що виконується, якщо не було винятків finally:

#Код блоку, що виконується в будь-якому випадку, можливо після відповідного блоку except

Для прикладу напишемо програму, в якій для введеного числа X буде знаходитися частка 1/X. Без опрацювання винятків дана програма буде мати наступний вигляд:

x=int(input()) k=1/x print(k)

Аналізуючи виконувані операції, можна зазначити дві з них, в яких можуть виникнути винятки:

* x=int(input()) – якщо користувач введе не ціле число, то функція int() не зможе введене значення привести до цілого і виникне виняток ValueError.
* k=1/x – якщо користувач введе 0, то інтерпретатору не вдасться виконати ділення на 0, і виникне виняток ZeroDivisionError. Доповнивши програму блоком опрацювання винятку, отримаємо: try:

x=int(input()) k=1/x

except ValueError:

print('Помилка: очікувалося ціле число.') except ZeroDivisionError:

print('Помилка: ділення на ноль.') else:

print(k)

Тобто при виникненні винятку буде виведене відповідне повідомлення про помилку.

**Зв’язування винятку зі змінною.**

Бувають ситуації, коли при обробці винятків достатньо виведення системного опису винятку. В такому випадку немає сенсу обробляти окремі винятки, достатньо обробити виняток Exception, але при цьому повернути опис винятку, що виник і є нащадком Exception. Для цього виняток, що виник, можна зв’язати зі змінною, використовуючи оператор as:

except Exception as <ім’я змінної>

Надалі в середині відповідного блоку except можна буде звернутися до змінної і отримати дані про виняток.

Наприклад, нашу попередню задачу перепишемо:

try:

x=int(input()) k=1/x except Exception as mes:

print('Помилка', type(mes),':', mes) else:

print(k)

Якщо користувач введе не ціле число (наприклад, 33.4), то отримає повідомлення:

Помилка <class 'ValueError'> : invalid literal for int() with base 10: '33.4'

Якщо користувач введе 0, то отримає повідомлення:

Помилка <class 'ZeroDivisionError'> : division by zero

## Виклик винятків

Оператор raise дозволяє програмісту згенерувати вказаний виняток. В єдиному аргументі raise вказується виняток, який буде викликано, наприклад:

>>> raise ZeroDivisionError('Ділення на нуль.') Traceback (most recent call last): File "<pyshell#9>", line 1, in <module> raise ZeroDivisionError('Ділення на нуль') ZeroDivisionError: Ділення на нуль