## Použití výjimek

V Pythonu se výjimky používají k signalizaci chyby nebo neočekávaného stavu během provádění kódu. Pokud dojde k výjimce, program se zastaví a vyhodí se chybová zpráva(hlášku), která popisuje, co se stalo. Výjimky se vyvolávají pomocí klíčového slova raise.

Využívá se především v try except blocích, když chceme vyhodit výjimku, ale stále chceme, aby kód dále pokračoval. Nebo je vyhazujeme, protože nechceme, aby program bez „něčeho“ stále fungoval. Nebo chceme ho vyhodíme a necháme na nějaké jiné části kódu, aby se o to postarala.

Následující příklad ukazuje, jak se vyvolá výjimka ValueError, pokud se zadaný argument nejedná o celé číslo:

def divide\_by\_two(num):

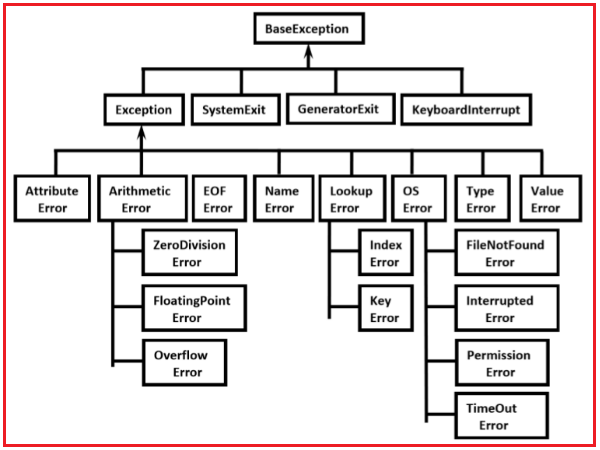
if not isinstance(num, int):

raise ValueError("Argument must be an integer")

return num / 2

print(divide\_by\_two(4)) # 2.0

print(divide\_by\_two("four")) # ValueError: Argument must be an integer



## Použití asercí

Aserce jsou podobné výjimkám, ale používají se pro kontrolu podmínek během provádění kódu. Pokud podmínka není splněna, vyvolá se výjimka AssertionError.

Asserci využíváme, když potřebujeme zastavit script z důvodu nějaké podmínky a něco vrátit pro lepší debuggování. Aserce méně pravděpodobněji zůstane ve výsledném programu na rozdíl od raise výjimek.

Následující příklad ukazuje, jak se používají aserce k ověření, zda je hodnota zadaného argumentu větší nebo rovna nule:

def square\_root(num):

assert num >= 0, "Number must be non-negative"

return num \*\* 0.5

print(square\_root(4)) # 2.0

print(square\_root(-4)) # AssertionError: Number must be non-negative

## Debuggování a zpracování chyb

Debuggování je proces hledání a opravování chyb v kódu. Python poskytuje několik nástrojů pro debuggování, včetně modulu pdb (Python Debugger). Navíc mnoho ide jako je VS code nebo přímo Pycharm nám dovoluje spustit proces debuggování kde můžeme procházet kód řádek po řádku.

import pdb

def factorial(n):

if n == 0:

return 1

else:

return n \* factorial(n-1)

pdb.set\_trace()

print(factorial(5))

následně pomocí příkazů, step, (next), p n … n je proměná která nás zajímá, list, help, print(content)

## Zpracování výjimek

Zpracování chyb je proces, kterým se zabýváme, když se v programu vyskytne výjimka. Při použití výjimek v Pythonu může programátor určit, co se má stát, pokud dojde k výjimce. Používá se blok try-except, který odchytne výjimku a provede nějaký kód, pokud k ní dojde. Používáme kód pro zpracování výjimek k odchycení chyb a provedení určitých kroků, aby se program nezastavil a nepřestal fungovat.

Následující příklad ukazuje, jak použít blok try-except, final a else k odchycení výjimky a provedení alternativního kódu:

**try**:

result = x / y

**except** **ZeroDivisionError**:

print("division by zero!")#vyskoci chyba

**else**:

print("result is", result)#nevyskoci chyba

**finally**:

print("executing finally clause")#probehne vzdy

pokud chceme využít více except bloků s nebo bez defaultní expect blok můžeme učinit takto (na co nesmíme zapomenout je, že defaultní except blok bez parametrů jde vždy až jako poslední)

**import** **sys**

**try**:

f = open('myfile.txt')

s = f.readline()

i = int(s.strip())

**except** **OSError** **as** err:

print("OS error:", err)

**except** **ValueError**:

print("Could not convert data to an integer.")

**except** **Exception** **as** err:

print(f"Unexpected **{**err**=}**, **{**type(err)**=}**")

**except**:

print(f"Some other error")

## Exception Group

**def** f():

**...**  excs = [**OSError**('error 1'), **SystemError**('error 2')]

**...**  **raise** ExceptionGroup('there were problems', excs)

f()

+ Exception Group Traceback (most recent call last):

| File "<stdin>", line 1, in <module>

| File "<stdin>", line 3, in f

| ExceptionGroup: there were problems

+-+---------------- 1 ----------------

| OSError: error 1

+---------------- 2 ----------------

| SystemError: error 2

+------------------------------------

## Custom Exception:

**class** SalaryNotInRangeError(Exception):

    """Exception raised for errors in the input salary.

    Attributes:

        salary -- input salary which caused the error

        message -- explanation of the error

    """

**def** \_\_init\_\_(self, salary, message="Salary is not in (5000, 15000) range"):

        self.salary = salary

        self.message = message

        super().\_\_init\_\_(self.message)

salary = int(input("Enter salary amount: "))

if not 5000 < salary < 15000:

    raise SalaryNotInRangeError(salary)

V Pythonu je třída BaseException základní třídou pro všechny výjimky, od kterých jsou ostatní výjimky odvozeny. Tato třída definuje několik základních atributů, které jsou důležité pro správné zacházení s výjimkami. Zde jsou tyto atributy:

* args: Tupple obsahující argumenty výjimky. Tyto argumenty jsou předány konstruktoru výjimky a lze je použít k zobrazení detailů o chybě.
* with\_traceback(tb): Metoda, která přiřadí zásobník volání traceback k výjimce. Pokud není poskytnuto žádné argument, použije se aktuální traceback.
* \_\_cause\_\_: Odkazuje na další výjimku, která způsobila tuto výjimku. Pokud tato výjimka není důsledkem jiné výjimky, bude hodnota tohoto atributu None.
* \_\_context\_\_: Odkazuje na kontext této výjimky, například chybu, která ji předcházela. Pokud tato výjimka není v kontextu žádné jiné výjimky, bude hodnota tohoto atributu None.
* \_\_traceback\_\_: Zásobník volání traceback, který byl přiřazen pomocí metody with\_traceback(). Pokud nebyla použita metoda with\_traceback(), bude hodnota tohoto atributu None.