Python

Třídy, objekty a výjimky

VŠCHT

2019

Deklarace tříd

Deklarace tříd - zásady

- deklarace třídy začíná klíčovým slovem *class* následovaným jménem třídy a dvojtečkou, deklarace funkce začíná klíčovým slovem *def*
- následují deklarace proměnných a funkcí třídy
- proměnné třídy jsou v jejím rámci dostupné přes klíčové slovo *self* které reprezentuje instanci této třídy

```
class UselessClass:
    useless_variable = "I can not do so much"

    def function(self):
        print(self.useless_variable)

my_useless_object = UselessClass()
print(my_useless_object.useless_variable)
my_useless_object.function()
```

Deklarace tříd

Deklarace tříd - zásady

- každá třída by měla mít funkci *__init()__* která se automaticky zavolá při vytváření její instance
- při vytvoření instance třídy lze předat parametry, ty jsou uvedeny v závorce metody *__init()__*
- pro navrácení hodnoty funkcí lze použít klíčové slovo return

```
class Person:
    def __init__(self, name, age):
        self.name = name
        self.age = age
    def tell_me_name(self):
        print("My name is ", self.name)
    def tell_me_age(self):
        return self.age

my_person = Person("John Doe", 50)
my_person.tell_me_name()  # prints My name is John Doe
print(my_person.tell_me_age())  # prints 50
```

Deklarace tříd

Objekty - mazání vlastnosti, objektů

- hodnoty vlastností objektů lze měnit po jejich vytvoření
- vlastnost objektu lze smazat
- objekt lze smazat

```
class Person:
    def __init__(self, name, age):
        self.name = name
        self.age = age
    def tell_me_name(self):
        print("My name is ", self.name)
    def tell_me_age(self):
        return self.age

my_person = Person("John Doe", 50)
my_person.tell_me_name()  # prints My name is John Doe
my_person.age = 7
print(my_person.tell_me_age())  # prints 7
del my_person.name  # deletes property
my_person.tell_me_name()  # error: AttributeError: 'Person' object has no attribute 'name'
del my_person  # delete object
my_person.tell_me_name()  # error: NameError: name 'my_person' is not defined
```

Dědění

Dědění - zásady

- rodiče lze předat jako parametr přímo třídě, funkce *super*() automaticky podědí všechny metody a vlastnosti rodiče
- pokud bude název metody potomka stejný jako název metody předka, bude metoda předka překryta

```
class Person:
    def __init__(self, name, age):
        self.name = name
        self.age = age
    def tell_me_name(self):
        print("My name is ", self.name)
    def tell_me_age(self):
        return self.age

class SuperHero(Person):
    def __init__(self, name, age, superpower):
        super().__init__(name, age)  # same as Person.init(self, name, age)
        self.superpower = superpower
    def tell_me_superpower(self):
        print(self.superpower)

my_hero = SuperHero("Aquaman", 33, "Hydrokinesis")
my_hero.tell_me_superpower()
my_hero.tell_me_name()
```

Výjimky

Chyby syntaxe

- před začátkem provádění programu se provádí kontrola jeho syntaktické správnosti
- při nalezení první chyby syntaxe je vyhlášen *SyntaxError* a do konzole se vypíše část kódu ve kterém byla nalezena chyba
- syntaktické chyby jsou nejčastější chyby začínajících programátorů

```
while True print("do nothing")
```

po spuštění do konzole vypíše následující hlášení, ve kterém je specifikováno číslo řádku a umístění syntaktické chyby

```
  File "C:/Users/[redacted]/PycharmProjects/vscht algoritmizece/syntax error.py", line 1
    while True print("do nothing")
                     ^
SyntaxError: invalid syntax
```

Výjimky

Typy výjimek

- Python obsahuje celou výjimek, více viz `https://docs.python.org/3/library/exceptions.html#bltin-exceptions`
- *ZeroDivisionError* - vznikne při dělení nulou
- *NameError* - vznikne pokud není lokální nebo globální jméno v kontextu programu nalezeno
- *TypeError* - vznikne při pokusu o operaci s objektem, který danou operaci neumožňuje
- *ValueError* - vznikne pokud operace nebo funkce obdrží argument správného typu ale špatné hodnoty

```
int("nonsense")  # ValueError
3 / "nonsense"  # TypeError
print(nonsense)  # NameError
1/0  # ZeroDivisionError
```

Výjimky - ošetření

Odchytávání výjimek

- každá část kódu kde očekáváme vznik výjimky by měla být ošetřena a příslušné možné výjimky odchyceny
- k tomu slouží příkazy *try*, *except*, *else* a *finally*
- příkaz *except* definuje blok kódu který se vykoná při vzniku výjimky
- příkaz *else* definuje blok kódu který se vykoná pokud nedošlo k žádné chybě v bloku *try*
- blok *finally* se vykoná vždy

Pozor

- vznik výjimky by neměl způsobit ukončení programu
- ošetření výjimek pomocí výpisu do konzole je krajně nevhodný, v následujících příkladech je takové ošetření pouze ilustrační
- v Pythonu je možné deklarovat si vlastní výjimky a třídy pro jejich ošetření, pro zájemce viz dokumentace jazyka Python

Výjimky - ošetření - příklad

```
def my_divide(number):
    try:
        print(23 / int(number))
    except ValueError:
        print("argument is not number")
    except ZeroDivisionError:
        print("number is 0, not possible to divide")
    except:
        print("something really unexpected went wrong")
    else:
        print("dividing done")
    finally:
        print("try except block passed")

my_divide(0)
my_divide("1")
my_divide("asfd")
```