Datové typy, Generika, Výčtové datové typy, Struktury, Anotace, Operátory

Datové typy
Datový typ je způsob definování významu a obsahu hodnoty proměnné nebo konstanty. Každý datový typ ma různé funkce, význam a operace, které se s nimi dají provádět. Značíme malým písmenem. Datové typy se dělí na dvě hlavní skupiny = primitivní a neprimitivní. V pythonu je tzv. dynamické typování = typ proměnné se určuje automatiky podle přiřazené hodnoty.

Primitivní datové typy
Jsou to takové datové typy, které mají definovanou velikost a paměťovou kapacitu, immutable, ukládají jednoduché hodnoty

byte = datový typ, který uchovává číslo v rozpětí mezi -128 do 127, paměťová kapacita je 8 bitů, základně je nastaven na 0

short = datový typ, který uchovává číslo v rozpětí -32 768 do 32 767, paměťová kapacita je 16 bitů, základně je nastaven na 0

int = uchovává číslo v rozpětí -2 147 483 648 do 2 147 483 647, paměťová kapacita je 32 bitů, celá čísla

float = datový typ, který uchovává desetinná čísla, 6-7 desetinných míst, paměťová kapacita je 32 bitů

double = datový typ, který uchovává desetinná čísla, 15 desetinných míst, paměťová kapacita je 64 bitů

boolean = datový typ, který uchovává pouze 2 informace - true/false, v paměti zapsán 0 nebo 1, paměťová kapacita je pouze 1 bit

Python nemá typy byte, short nebo double jako samostatné datové typy. Místo toho používá modul `struct` nebo knihovny jako NumPy, pokud potřebujeme pracovat s konkrétními bitovými velikostmi.

Neprimitivní datové typy
Říká se jim ještě referencové, jsou to takové datové typy, které ukládají kolekce hodnot nebo složitější struktury, mutable, nemají pevně danou paměťovou kapacitu, proměnné těchto typů obsahují reference, nikoli samotné hodnoty

List = objekt, který nám umožňuje do něj ukládat elementy různých typů, objekty uložené do pole jsou ukládány vedle sebe v paměti, indexované, funkce len()
list = [1231,412,41,412,4,5432,]

Slovník = ukládá páry klíč-hodnota, optimalizovaná hashovací tabulka - rychlý přistup k datům, jeho velikost dynamicky roste
dict = {“jmeno” : “Jan”, “vek” : 19}

Set = ukládá neuspořádanou kolekci unikátních prvků, odstraňuje duplitcity
muj_set = {1,2,3,4,5,6}

Tuple = uspořádaná sekvence hodnot, neměnné
muj_tuple = (1,3,2,5,7)

Class = uživatelsky definovaná datová struktura s atributy a metodami

```
class Student:
    def __init__(self, jmeno, vek):
        self.jmeno = jmeno
        self.vek = vek
        self.predmety = []

    def pridej_predmet(self, predmet):
        self.predmety.append(predmet)

    def __str__(self):
        return f"Student {self.jmeno}, věk: {self.vek}"

# Vytvoření instance
jan = Student("Jan Novák", 19)
jan.pridej_predmet("Matematika")
print(jan)  # "Student Jan Novák, věk: 19"
```

Generika
Generika nám umožňují do určitých kolekcí vkládat prvky stejného datového typu (pouze int, string, objekty trídy….). Slouží k lepší čitelnosti kódu, dokumentaci a umožňují statickou typovou kontrolu pomocí nástrojů jako je mypy. Jelikož je python dynamicky typovaný jazyk, generika se v něm nevyuživají podobně jako třeba u C# nebo Javy.

V pythonu se používají tzv. typové anotace. Je to způsob jak naznačit typ proměnných, parametrů funkcí a returnů. Nejsou to úplně pravá generika

```
# Základní typové anotace
jmeno: str = "Jan"
vek: int = 19
vyska: float = 1.85
```

```
je_student: bool = True

# Typová anotace funkce
def pozdrav(jmeno: str, vek: int) -> str:
    return f"Ahoj {jmeno}, je ti {vek} let."

# Volání funkce
vysledek = pozdrav("Jan", 19)
```

Při použití složitějších typů je důležité nezapomenout na to je importovat!!!

```
from typing import List, Dict, Tuple, Optional

# Seznam čísel
cisla: List[int] = [1, 2, 3, 4]

# Slovník - jméno -> věk
vek_studentu: Dict[str, int] = {"Jan": 19, "Petr": 20}

# Tuple - jméno, věk, průměr
student_info: Tuple[str, int, float] = ("Jan", 19, 1.5)

# Optional - může být řetězec nebo None
prijmeni: Optional[str] = None
```

Výčtové datové typy

Enum, speciální datový typ, který umožňuje definovat pojmenované konstanty. Výčtový typ představuje množinu hodnot, které může proměnná tohoto typu nabývat. Jde o způsob, jak vytvořit sadu pojmenovaných konstant, které patří logicky k sobě. Výhodami je čitelnost kódu, omezení chyb, seskupení souvisejících konstant

```
from enum import Enum, auto

# Definice enum třídy pro priority úkolů
class Priorita(Enum):
    NIZKA = 1
    STREDNI = 2
    VYSOKA = 3

    def je_dulezite(self):
        # Metoda určující, zda je úkol důležitý
        return self.value >= 2

# Příklad použití
```

```
muj_ukol_priorita = Priorita.VYSOKA

# Přístup k hodnotám
print(f"Priorita úkolu: {muj_ukol_priorita.name}")  # VYSOKA
print(f"Číselná hodnota: {muj_ukol_priorita.value}")  # 3

# Použití vlastní metody
if muj_ukol_priorita.je_dulezite():
    print("Úkol je důležitý!")
```

Struktury

Struktury v pythonu jsou implementovány jako třídy. Umožňují vytvářet vlastní datové typy, které kombinují data a funkce. Mohou obsahovat různé datové typy. Alokují se na haldě. Python oproti jiným jazykům, funguje tak,že vše je objekt. Pomáhají udržovat kód strukturovaným. OOP

```
# Nejjednodušší struktura v Pythonu (třída jen s daty)
class Osoba:
    def __init__(self, jmeno, vek):
        self.jmeno = jmeno
        self.vek = vek

# Vytvoření a použití
student = Osoba("Jan Novák", 19)
print(f"{student.jmeno} je {student.vek} let starý")
```

Anotace (dekorátory)

Anotace jsou doprovodné informace (metadata). Označují se pomocí @ před jménem. Zpracovávají se při běhu programu. Umožňují modifikovat funkce nebo třídy bez změny v jejich kódu. Dávají nám možnost přidat funkcionalitu před/po provedení funkce

Vestavěné dekorátory
@staticmethod - označují metodu, která nepracuje s instancí
@classmethod - pracuje s třídou místo instance
@property - převádí metodu na atribut

```
class Student:
    # Třídní proměnná
    pocet_studentu = 0

    def __init__(self, jmeno):
        self.jmeno = jmeno
```

```
        self._znamky = []  # "soukromý" atribut
        Student.pocet_studentu += 1

    # Přístup k metodě bez volání instance
    @staticmethod
    def o_skole():
        print("Toto je třída pro studenty gymnázia.")

    # Přístup ke třídním proměnným
    @classmethod
    def kolik_studentu(cls):
        return f"Počet studentů: {cls.pocet_studentu}"

    # Metoda použitelná jako atribut (bez závorek)
    @property
    def prumer(self):
        if not self._znamky:
            return None
        return sum(self._znamky) / len(self._znamky)

    # Nastavení hodnoty pro property
    @prumer.setter
    def prumer(self, hodnota):
        raise ValueError("Průměr nelze nastavit přímo!")

# Použití
jan = Student("Jan")
Student.o_skole()  # Volání statické metody
print(Student.kolik_studentu())  # Volání třídní metody
print(jan.prumer)  # Volání property (bez závorek)
```

Operátory

Speciální symboly, které vykonávají operaci nad hodnotami - matematické výpočty, porovnávání, logické operace a manipulace s daty. Máme několik druhů - Aritmetické, Porovnávající, Logické a Bitové

Operators in Python

Operators	Type
+, -, *, /, %	Arithmetic operator
<, <=, >, >=, ==, !=	Relational operator
AND, OR, NOT	Logical operator
&, |, <<, >>, -, ^	Bitwise operator
=, +=, -=, *=, %=	Assignment operator