# I01

**Datové typy a operace s daty. Funkce. Abstraktní datový typ, zásobník, fronta. (Základy programování)**

## Datové typy

Určuje v programování rozsah a druh hodnot, kterých nabývá proměnná. Tedy dá se říct, že určují význam čísel, které jsou uloženy v paměti. Datový typ je omezen oborem hodnot a zjednodušeně se jedná o to, jaké typy hodnot lze do jednotlivých proměnných přiřazovat. Zároveň mi každý datový typ definuje (určuje), jaké operace s ním lze a nelze provádět, tedy například že nelze od sebe odečíst dvě proměnné obsahující řetězec znaků.

Datové typy můžeme deklarovat:

* **explicitně** - přímo uvedu jakého datového typu daná proměnná je, viz. C, int x;
* **implicitně** - datový typ je určen automaticky kompilátorem (Python je kompilovaný jazyk)

Typová kontrola je v programování kontrola datového typu proměnný:

* **staticky** - na začátku je zkontrolováno kompilátorem, zda všechny proměnné mají správný datový typ a jestli operace s proměnnými jsou správný (př. C, kdy se typová kontrola provede kompilátorem)
* **dynamicky** - typová kontrola je prováděna za běhu programu a je možno i za běhu programu měnit datový typ podle potřeby

Python využívá dynamické implicitní typování, to znamená, že datový typ se určuje automaticky, není třeba ho zadávat. Může se měnit za běhu.

**Přetypování** je proces, kdy dochází ke změně datového typu proměnný.

Python - int(x), C - (int) x;, java - (int)

**Deklarace** - přiřazení jména (identifikátoru) a typ nějakému objektu jazyka, nejčastěji proměnné, funkci, konstantě atd.

**Datové typy Python:**

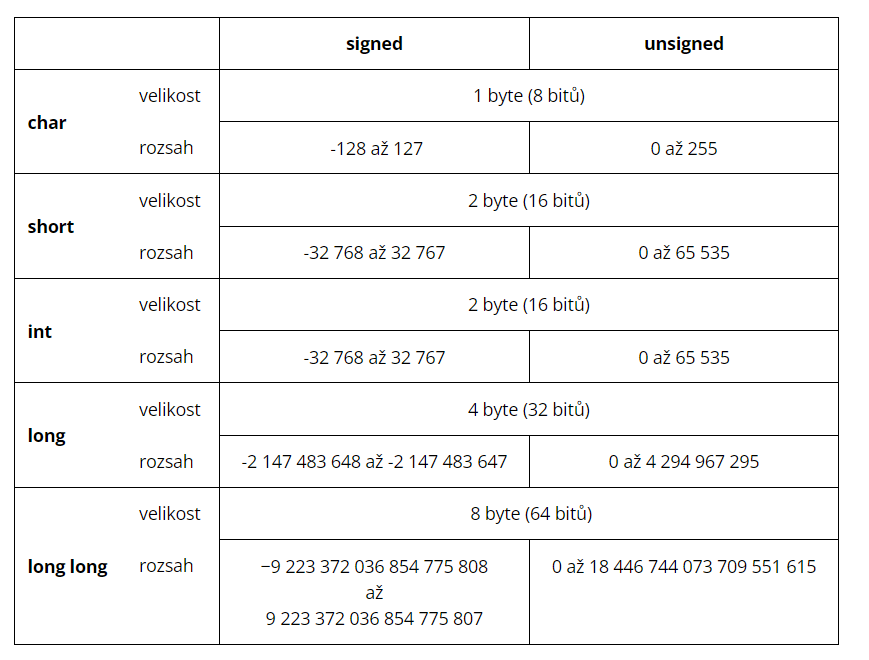
* **Primitivní datové typy**

1. Prázdný typ - None
2. celé číslo - int
3. racionální číslo - float
4. komplexní číslo - komplex
5. řetězec - string

* **Strukturované datové typy**

1. seznam - list (na libovolném místě může být libovolný prvek s libovolným datovým typem )
2. n-tice - tuple
3. množina - set
4. slovník - dict

**Datové typy C:**

****

String - pole charů

Enum - výčtový typ

Array - pole, všechny prvky pole musí mít stejný datový typ

## Operace s daty

S datovými objekty lze provádět různé operace. To jest nějak s nimi zacházet pomocí operátorů. Operace může být například matematická operace sečtení dvou čísel, kde jako operátor funguje znaménko + (plus). Ty dvě čísla pak nazýváme operandy. Každá operace vrací nějaký výsledek.

Operace ale nejsou jen matematické, je to například i porovnání dvou operandů na shodu nebo identitu a řada dalších. Python podporuje spoustu operátorů a podrobně se jim věnuji v kapitole [operátory](http://python.wraith.cz/zaklady-operatory.php).

## Funkce

Funkce je pojmenovaný blok kódu, na který se můžeme odkázat v jiné části programu a vykonat tak kód, který se ve funkci nachází. Slouží pro strukturování kódu anebo když určitá část kódu je využívaná vícekrát, tak se napíše do funkce, která se může libovolně volat.

U funkcí rozlišujeme určité termíny:

* **parametry** - Jsou to proměnné, které jsou dostupné uvnitř funkce, jejichž hodnotu nastavujeme při zavolání dané funkce.
* **argument** - hodnoty (výrazy), které se předávají do funkce, když je volaná
* **návratová hodnota** - hodnota, kterou funkce vrací
* **definice** - vytvoření funkce(v c musí být funkce nejdříve deklarována, pokud je definována na začátku, je zároveň i deklarována)

**funkce v C**

<datový typ návratové hodnoty funkce> *název\_funkce* (parametry funkce){

// *kód funkce*

return hodnota;

}

**funkce v Pythonu**

def *název\_funkce* (parametry funkce):

// kód

## Abstraktní datové typy

**Datový typ** obecně popisuje rozsah hodnot, které patří do daného typu a operace, které je možno s těmito hodnotami provádět.

**Abstraktní datový typ** popisuje rozhraní, které je možno s tím objektem provádět. Nezajímá mě až tolik ta implementace.

Zásobník, fronta, množina,seznam

Je to datová struktura, která má definované operaci pro práci s touto datovou strukturou.

Dva pohledy na data:

* **Abstraktní**

Zajímají nás operace, které s daty budu provádět. Ale nezajímá mě, jak je to neimplementované uvnitř. Např. ulož, najdi, vymaž

* **Implementační**

Zajímá mě jak jsou data uloženy, jak jsou jednotlivé operace implementovány atd. Prostě mě zajímá vnitřní struktura.

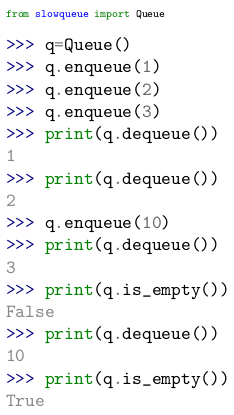
## Abstraktní datová struktura

Abstraktní datová struktura je způsob, jak efektivně uložit data tak, aby práce s nimi byla relativně snadná. Je to abstraktní skladiště pro data definovaná v rámci množiny operací a pro výpočetní složitosti při vykonávání těchto operací, bez ohledu na implementaci v konkrétní datové struktuře.

Výběr abstraktní datové struktury je rozhodující pro design účinných algoritmů a pro odhad jejich složitosti, zatímco výběr konkrétních datových struktur je důležitý pro účinnou implementaci těchto algoritmů.

Tento pojem je velmi blízký pojmu abstraktní datový typ, který se používá v teorii programovacích jazyků. Názvy mnoha abstraktních datových struktur (a abstraktních datových typů) odpovídají názvům konkrétních datových struktur.

**Zásobník** (Stack)

* strukturovaný/složený datový typ
* obsahuje předem neznámé množství položek, typicky stejného typu (jako pole)
* podporuje následující operace (se složitostí O(1))
  + přidání položky na konec (**push**)
  + odebrání položky z konce (**pop**)
  + test, jestli je zásobník prázdný (**is\_empty**)
* položky jsou odebírány v opačném pořadí, než byly přidány, (**LIFO** — last in first out)
* zásobník může podporovat i další operace
  + nedestruktivní čtení z konce (**peek**)
  + zjištění počtu položek na zásobníku (**size**)
* využití pro dočasné ukládání dat, např. stavu algoritmu

**Fronta** (Queue)

* strukturovaný/složený datový typ
* obsahuje předem neznámé množství položek, typicky stejného typu (jako pole)
* podporuje následující operace (se složitostí O(1))
  + přidání položky na konec (**enqueue**, **add**)
  + odebrání položky ze začátku (**dequeue**, **top**)
  + test, jestli je fronta prázdná (**is\_empty**)
* položky jsou odebírány ve stejném pořadí, jako byly přidány, (**FIFO** — first in first out)
* fronta může podporovat i další operace
  + nedestruktivní čtení ze začátku (**peek**)
  + zjištění počtu položek ve frontě (**size**)

Fronta - použití

* Komunikace mezi procesy (consumer,producer)
* Čekání na asynchronní periferie — klávesnice, disk, síť . . .
* ‘Férový přístup’ pro sdílení zdrojů (policy)
* Simulace čekání ve frontě
* Některé grafové a třídící algoritmy (prohledávání do šířky, merge sort)

Fronta - pole s realokací

* Prvky ukládáme do pole
* Vkládání na konec pole (O(1))
* Prvky nemažeme, ale posouváme index počátku
* Pokud je vynechaných prvků hodně, překopírujeme frontu do nového pole
* Kopírujeme po poklesu využití paměti na 50 %
* Odebrání n prvků vyžaduje ∼ log2n kopírování, dohromady n/2 + n/4 + · · · ∼ n prvků → amortizovaná složitost odebrání prvku je O(1)
* Nevýhoda — neefektivní využití paměti

Fronta - dva zásobníky

* Prvky ukládáme do zásobníku inp (O(1))
* Prvky odebíráme ze zásobníku out (O(1))
* Když je out prázdný, přesuneme do něj inp
* Každý prvek je kopírován jen jednou, složitost zůstává O(1)