IB111 Úvod do programování skrze Python
Přednáška 6

Datové typy

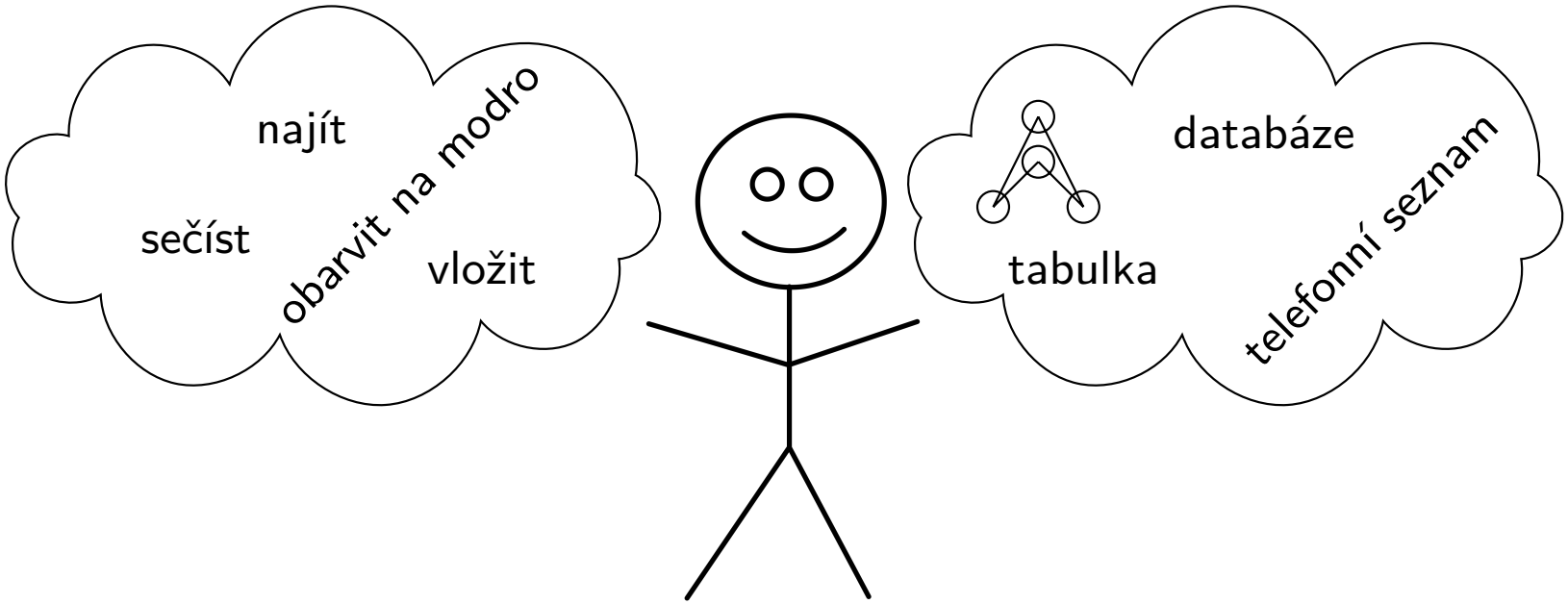
Nikola Beneš

26. říjen 2016

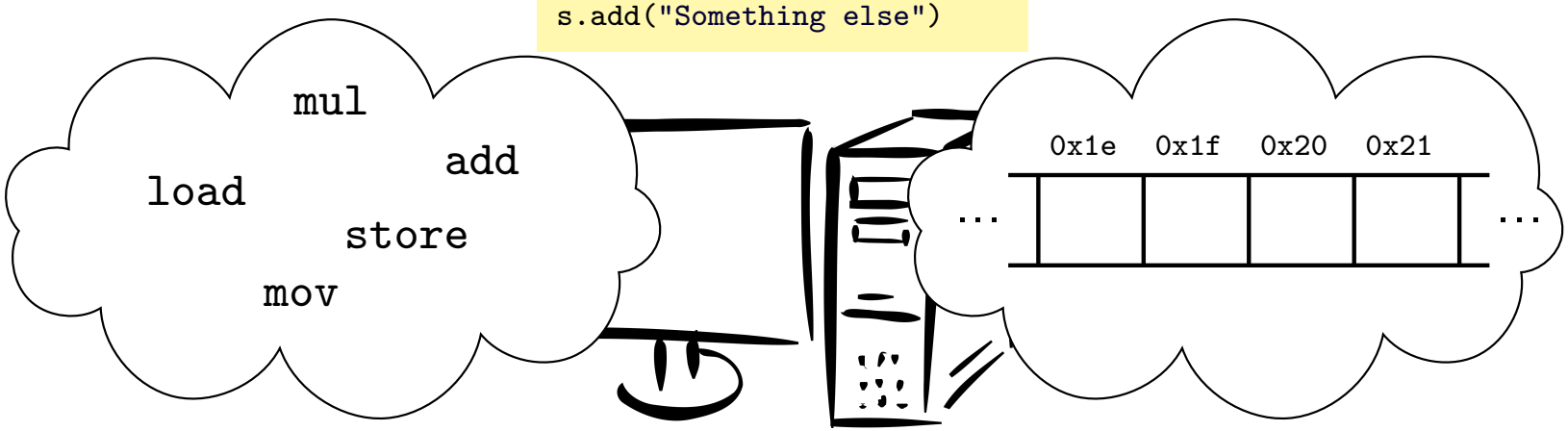
Práce s daty

- jaká data budu zpracovávat?
- jaká data budu potřebovat k řešení problému?
- jaké operace s daty budu chtít provádět?
- jak rychle budu chtít, aby operace s daty fungovaly?

Pohled na data – dva světy




```
s = set()
s.add("Something")
s.add("Something else")
```



Datové typy

Datový typ

- rozsah hodnot, které patří do daného typu
- operace, které je možno s těmito hodnotami provádět

Abstraktní datový typ

- rozhraní
- popis operací, které chceme provádět (případně i složitost)

Konkrétní datový typ (datová struktura)

- implementace
- přesný popis uložení dat v paměti
- definice funkcí pro práci s těmito daty

Poznámka: hranice mezi abstraktním a konkrétním datovým typem není vždy úplně ostrá.

Abstraktní datové typy

Nejznámější ADT:

- seznam
- zásobník
- fronta; prioritní fronta
- množina
- slovník (asociativní pole)

- textový řetězec
- celé číslo

... a mnohé další

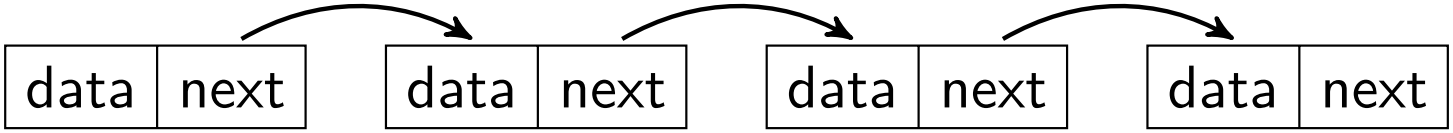
Datový typ seznam

Seznam (různé varianty)

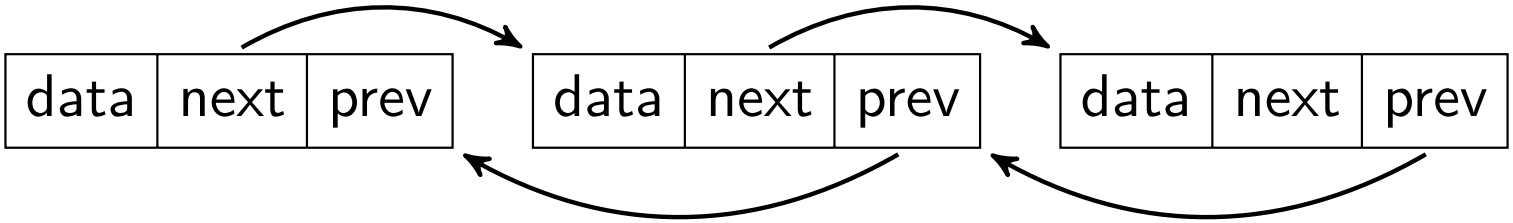
- obsahuje posloupnost prvků
  - stejného typu
  - různého typu
- přidání prvku
  - na začátek
  - na konec
  - na určené místo
- odebrání prvku
  - ze začátku
  - z konce
  - konkrétní prvek
- test prázdnosti
- a možná i další operace
  - např. přístup pomocí indexu

Implementace seznamu

Jednosměrně zřetězený seznam



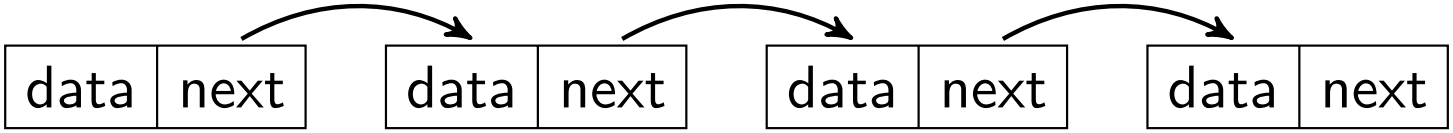
Obousměrně zřetězený seznam



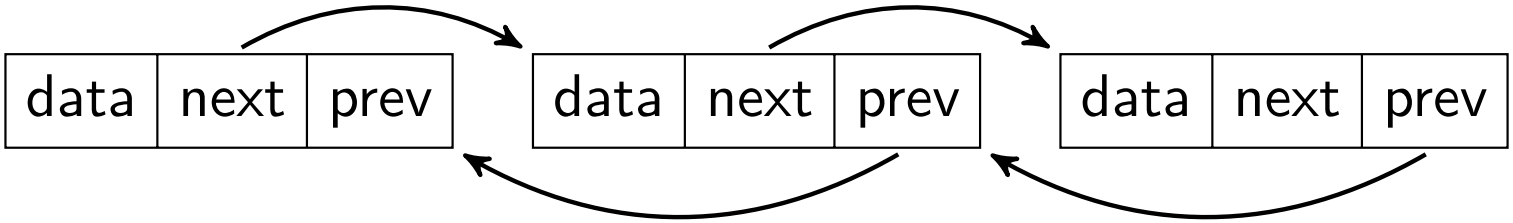
Dynamické pole

Implementace seznamu

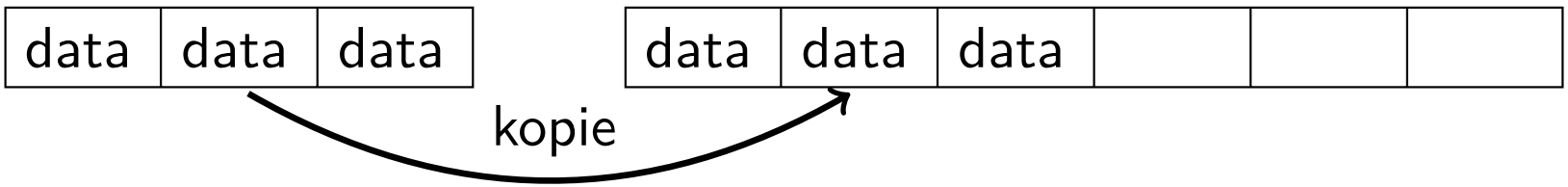
Jednosměrně zřetězený seznam



Obousměrně zřetězený seznam

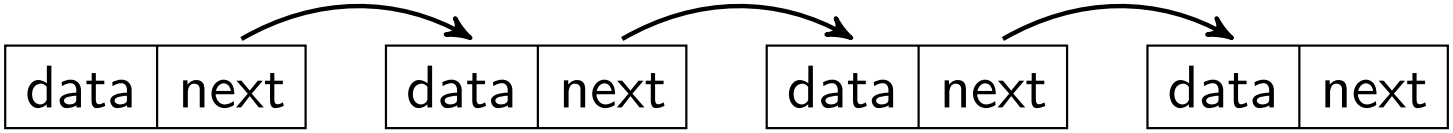


Dynamické pole

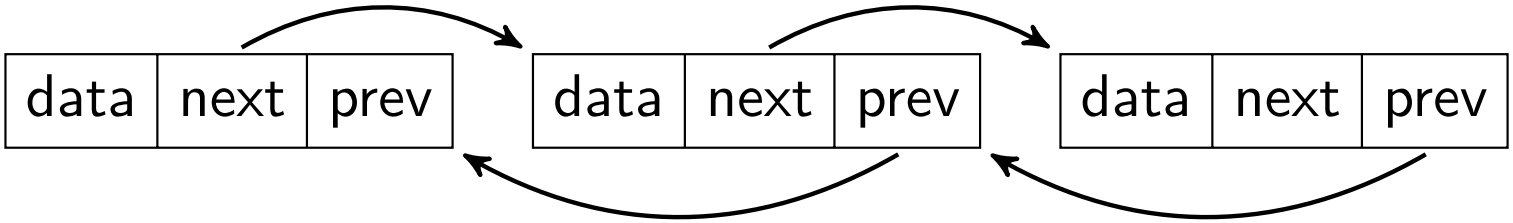


Implementace seznamu

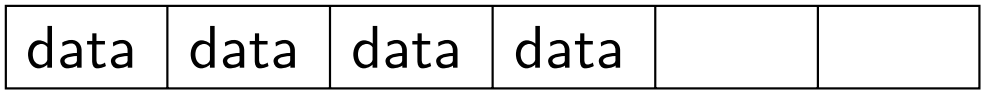
Jednosměrně zřetězený seznam



Obousměrně zřetězený seznam



Dynamické pole



Seznamy v Pythonu

Opakování

- hranaté závorky, prvky oddělené čárkami
- prvky mohou být různých typů
- přístup skrze indexy
  - indexování od konce pomocí záporných čísel
- seznamy lze modifikovat

```
a = ['bacon', 'eggs', 'spam', 42]
print(a[1:3])          # ['eggs', 'spam']
print(a[-2:-4:-1])     # ['spam', 'eggs']
a[-1] = 17
print(a)               # ['bacon', 'eggs', 'spam', 17]
print(len(a))          # 4
```

K zamyšlení: Jakou implementaci seznamu používá Python?

Seznamy v Pythonu (pokr.)

Užitečné funkce pro seznamy

```
l.append(x)      # přidá prvek x na konec
l.extend(s)      # přidá všechny prvky s na konec
l.insert(i, x)   # přidá prvek x před prvek na pozici i
l.remove(x)      # odstraní první prvek rovný x
l.pop(i)         # odstraní (a vrátí) prvek na pozici i
l.pop()          # odstraní (a vrátí) poslední prvek
l.index(x)       # vrátí index prvního prvku rovného x
l.count(x)       # vrátí počet výskytů prvků rovných x
l.sort()         # seřadí seznam
l.reverse()      # obrátí seznam
x in l           # test, zda seznam obsahuje x
                 # (lineární průchod seznamem!)
```

Seznamy v Pythonu (pokr.)

Příkaz `del`

- smaže prvek nebo část seznamu

```
a = ['spam', 'bacon', 'eggs', 'spam', 42]
del a[-1]
print(a)     # ['spam', 'bacon', 'eggs', 'spam']
del a[1:3]
print(a)     # ['spam', 'spam']
del a[:]
print(a)     # []
```

- `del` umí mazat i jiné věci, např. celé proměnné

```
del a
print(a)     # NameError: name 'a' is not defined
```

Použití seznamů

Příklad: medián

```
def median(num_list):
    num_list.sort()
    l = len(num_list)
    mid = l // 2
    if l % 2 != 0:
        return num_list[mid]
    return (num_list[mid] + num_list[mid - 1]) / 2

a = [6, 4, 4, 6, 5, 3, 6, 4, 2, 2]
print(median(a))      # 4.0
print(a)              # [2, 2, 3, 4, 4, 4, 5, 6, 6, 6]
```

- co se stalo a jak to spravit? (vedlejší efekt funkce)

Poznámka: medián se dá spočítat i bez seřazení seznamu

Generátorová notace pro seznamy (*list comprehension*)

- specialita Pythonu, vyskytuje se v některých jiných jazycích (Haskell)

```
s = [x for x in range(1, 7)]
print(s)    # [1, 2, 3, 4, 5, 6]

s = [2 * x for x in range(1, 7)]
print(s)    # [2, 4, 6, 8, 10, 12]

s = [(a, b) for a in range(1, 5) for b in ["A", "B"]]
print(s)    # [(1, 'A'), (1, 'B'), (2, 'A'), (2, 'B'), ...
```

- připomíná vám to něco?
  - zápis množin pomocí charakteristické vlastnosti prvků (*set comprehension*)

Vnořené seznamy

- prvky seznamů mohou být opět seznamy
- použití: vícerozměrná data (např. matice)

```
mat = [[1, 2, 3],
       [4, 5, 6],
       [7, 8, 9]]
print(mat[1][2])     # 6
```

Příklad: nulová matice zadaných rozměrů

```
def null_matrix(m, n):
    return [[0 for col in range(n)] for row in range(m)]
```

- lepší způsob práce s maticemi: knihovna `numpy`

Vnořené seznamy (pokr.)

Příklad: násobení matic

```
def matrix_mult(matL, matR):
    rows = len(matL)
    cols = len(matR[0])
    common = len(matR)
    result = null_matrix(rows, cols)
    for i in range(rows):
        for j in range(cols):
            for k in range(common):
                result[i][j] += matL[i][k] * matR[k][j]
    return result
```

K zamyšlení: jak ošetříme, že vstup je platný?

- jsou na vstupu skutečně matice?
- jsou matice kompatibilní?

Ntice (tuples) v Pythonu

- podobné seznamům
- neměnné jako řetězce
- zápis do kulatých závorek (nepovinné)
- specialita Pythonu: `(7,)` je ntice o velikosti 1

```
t = "The Answer", 42, 6 * 9
print(t)          # ('The Answer', 42, 54)
print(t[1])       # 42
u = t, (1, 2, 3)
print(u)          # (('The Answer', 42, 54), (1, 2, 3))
```

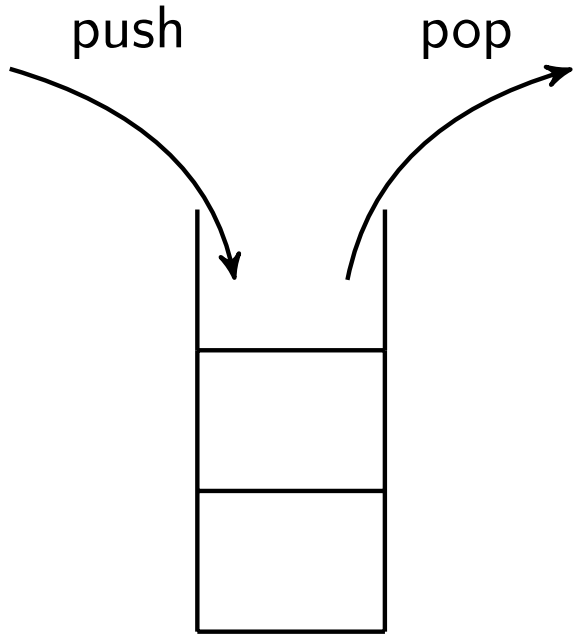
- použití při přiřazení

```
x, y, z = t       # přiřazení do x, y i z
print(z)          # 54
x, y = y, x       # prohození proměnných
```

Datový typ zásobník

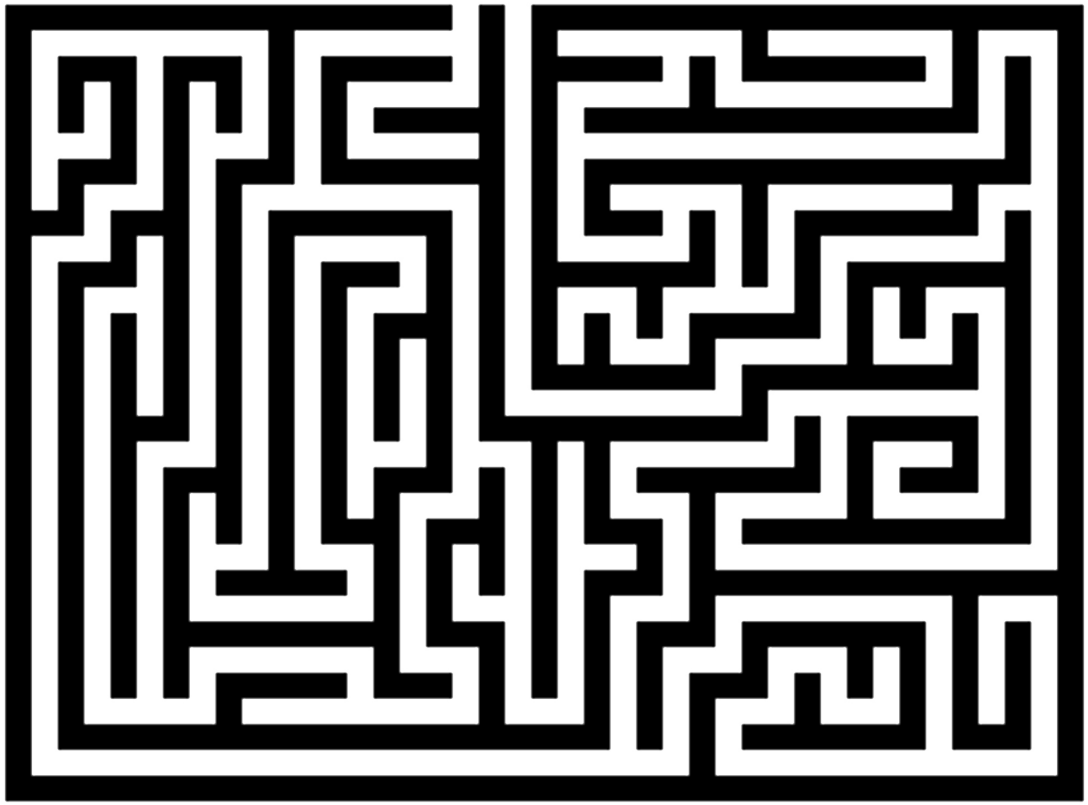
Zásobník

- obsahuje prvky v pořadí LIFO (*Last In First Out*)
- operace
  - push (vložení)
  - pop (odstranění)
  - top (náhled na horní prvek)
  - empty (test prázdnosti)
- mnohá použití
  - procházení grafů
  - analýza syntaxe
  - vyhodnocování výrazů
  - rekurze



Motivace pro zásobník

- procházení bludiště bez smyček



Zásobník v Pythonu

- implementace pomocí seznamu
  - místo push máme append
  - místo top máme `[-1]`

```
def push(stack, element):
    stack.append(element)

def pop(stack):
    return stack.pop()

def top(stack):
    return stack[-1]

def empty(stack):
    return stack.empty()
```

Zásobník v Pythonu (pokr.)

Příklad: postfixová notace

- infixová notace
  - operátory mezi operandy
  - např. 1 + 2, (3 + 7) * 9
  - je třeba používat závorky
- prefixová notace (polská notace)
  - operátory před operandy
  - např. + 1 2, * + 3 7 9
  - není třeba závorky
- postfixová notace (reverzní polská notace, RPN)
  - operátory za operandy
  - např. 1 2 +, 3 7 + 9 *
  - není třeba závorky
  - snadné vyhodnocení pomocí *zásobníku*

Zásobník v Pythonu (pokr.)

Příklad: postfixová notace

```
def eval_rpn(line):
    stack = []
    for token in line.split():
        if token == '*':
            b = pop(stack)
            a = pop(stack)
            push(stack, a * b)
        elif token == '+':
            b = pop(stack)
            a = pop(stack)
            push(stack, a + b)
        else:
            push(stack, float(token)
    return top(stack)
```

Zásobník v Pythonu (pokr.)

Příklad vyhodnocení postfixové notace: `7 4 7 + * 8 +`

vstup	akce	zásobník
`7 4 7 + * 8 +`	push	
`4 7 + * 8 +`	push	7
`7 + * 8 +`	push	7 4
`+ * 8 +`	+	7 4 7
`* 8 +`	*	7 11
`8 +`	push	77
`+`	+	77 8
		85

Datový typ fronta

Fronta

- obsahuje prvky v pořadí FIFO (*First In First Out*)
- operace
  - enqueue (vložení)
  - dequeue (odstranění)
  - front (náhled na přední prvek)
  - empty (test prázdnosti)
- použití
  - zpracovávání příchozích požadavků

Fronta v Pythonu

- implementace pomocí seznamů by byla pomalá
  - přidávání a odebírání na začátek seznamu vyžaduje přesun
- použití knihovny `collections`
  - datový typ `deque` (oboustranná fronta)
  - vložení do fronty pomocí `append`
  - odebrání z fronty pomocí `popleft`
  - přední prvek fronty je `[0]`

```
from collections import deque
q = deque(["Eric", "John", "Michael"])
q.append("Terry")    # Terry arrives
q.append("Graham")   # Graham arrives
q.popleft()          # Eric leaves
q.popleft()          # John leaves
print(q)             # deque(['Michael', 'Terry', 'Graham'])
```

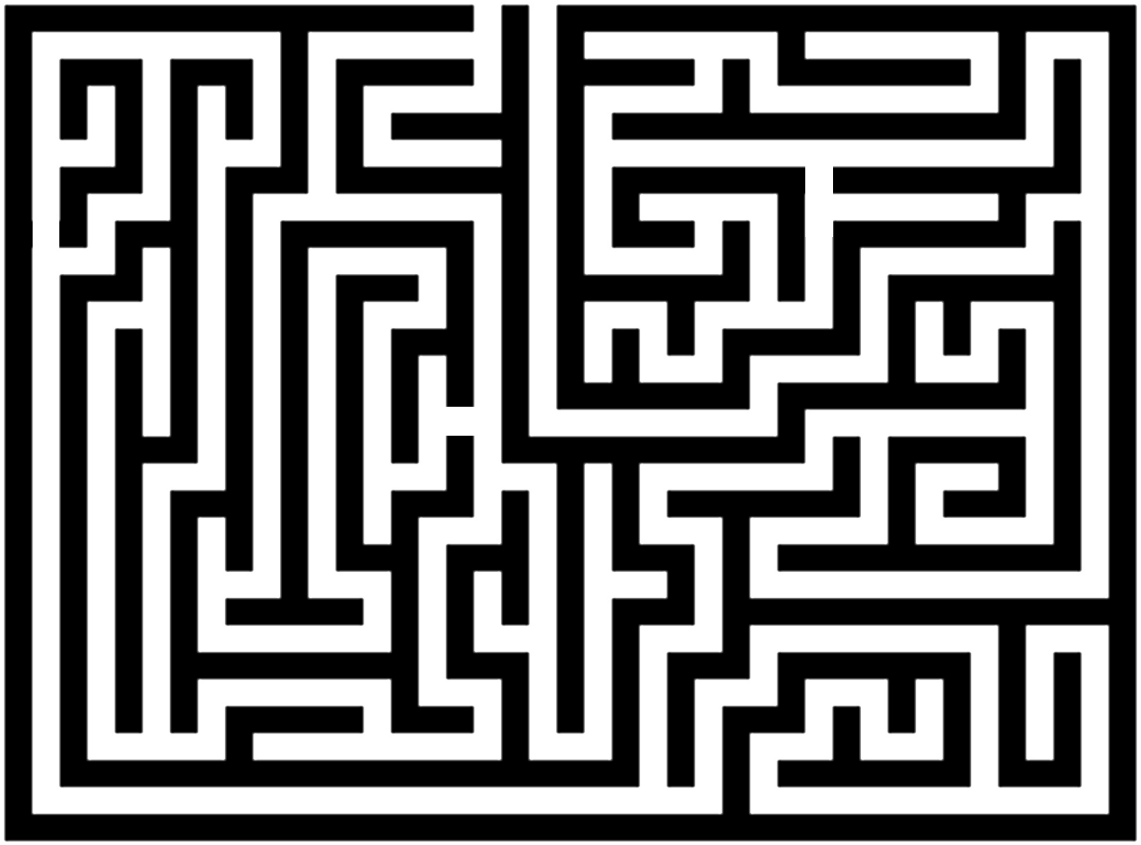
Datový typ množina

Množina

- neuspořádaná kolekce dat bez vícenásobných prvků
- operace
  - insert (vložení)
  - find (vyhledání prvku, test přítomnosti)
  - delete (odstranění)
- použití
  - grafové algoritmy (označení navštívených vrcholů)
  - rychlé vyhledávání
  - výpis unikátních slov

Motivace pro množinu

- procházení bludiště se smyčkami



Množina v Pythonu

- speciální datový typ set

```
set(l)                          # vytvoří množinu ze seznamu
len(s)                          # počet prvků množiny s
s.add(x)                        # přidání prvku do množiny
s.remove(x)                     # odebrání prvku z množiny
x in s                          # test, zda množina obsahuje x
s1 <= s2                        # test, zda je s1 podmnožinou s2
s1.union(s2)                    # sjednocení množin s1 a s2
s1 | s2                         # -- totéž --
s1.intersection(s2)             # průnik množin s1 a s2
s1 & s2                         # -- totéž --
s1.difference(s2)               # rozdíl množin s1 a s1
s1 - s2                         # -- totéž --
s1.symmetric_diference(s2)      # symetrický rozdíl množin s1 a s2
s1 ^ s2                         # -- totéž --
```

Množina v Pythonu (pokr.)

```
basket = ['apple', 'orange', 'apple', 'orange', 'banana']
fruit = set(basket)
print(fruit)                    # {'orange', 'apple', 'banana'}
print('orange' in fruit)  # True
print('tomato' in fruit)  # False

a = set("abracadabra")
b = set("engineering")
print(a)        # {'a', 'r', 'b', 'c', 'd'}
print(b)        # {'i', 'r', 'e', 'g', 'n'}
print(a | b)  # {'a', 'c', 'b', 'e', 'd', 'g', 'i', 'n', 'r'}
print(a & b)  # {'r'}
print(a - b)  # {'a', 'c', 'b', 'd'}
print(a ^ b)  # {'a', 'c', 'b', 'e', 'd', 'g', 'i', 'n'}
```

Datový typ slovník

Slovník (dictionary, map, asociativní pole)

- neuspořádaná množina dvojic (klíč, hodnota)
- klíče jsou unikátní
- operace jako u množiny (insert, find, delete)
- navíc přístup k hodnotě pomocí klíče
- klíče jsou neměnné, ale hodnoty se smí měnit
- použití
  - překlad UČO na jméno, jméno na tel. číslo apod.
  - počet výskytů slov v textu
  - „cache“ výsledků náročných výpočtů

Slovník v Pythonu

- zápis do složených závorek {}
- klíč a hodnotu oddělujeme dvojtečkou
- záznamy oddělujeme čárkami

```
phone = {"Buffy": 5550101, "Xander": 5550168}
phone["Dawn"] = 5550193
print(phone)
# {'Xander': 5550168, 'Dawn': 5550193, 'Buffy': 5550101}
print(phone["Xander"])
# 5550168
del phone["Buffy"]
print(phone)
# {'Xander': 5550168, 'Dawn': 5550193}
print(phone.keys())
# dict_keys(['Xander', 'Dawn'])
print("Dawn" in phone)
# True
```

Slovník v Pythonu (pokr.)

- procházení všech položek ve slovníku – `.items()`
  - nepoužívejte pro vyhledávání!

```
for name, num in phone.items():
    print(name + "'s number is", num)
# Xander's number is 5550168
# Dawn's number is 5550193
```

- užitečné funkce pro slovníky

```
d.items()                 # vrátí seznam záznamů (dvojic)
d.get(key, default)       # pokud existuje klíč key, vrátí jeho
                          # hodnotu, jinak vrátí hodnotu default
d.get(key)                # jako předtím,
                          # jen default je teď None
```

Slovník v Pythonu – příklady použití I

Frekvence slov

```
def is_word_char(char):
    return char not in '!"#$%&\'()*+,-./:;<=>?@[\\]^_`{|}~'

def word_freq(text):
    text = "".join(filter(is_word_char, text))
    text = text.lower()
    word_list = text.split()
    freq = {}
    for word in word_list:
        freq[word] = freq.get(word, 0) + 1
    return freq
```

Slovník v Pythonu – příklady použití I (pokr.)

Frekvence slov – výpis frekvencí

```
def output_word_freq(text):
    freq = sorted(word_freq(text).items(),
                  key=lambda x: (x[1], x[0]),
                reverse=True)
    print("Word frequencies, sorted from the most frequent:")
    for word, count in freq:
        print(word, "\t", count)
```

Slovník v Pythonu – příklady použití I (pokr.)

Frekvence slov

```
laf = """I must not fear. Fear is the mind-killer. ..."""
# etc.
output_word_freq(laf)
```

```
will      5
i         5
fear      5
the       4
to        2
me        2
...
```

Slovník v Pythonu – příklady použití II

Morseovka

```
morse = {'A': '.-', 'B': '-...', 'C': '-.-.'} # etc.

def to_morse(s):
    result = ""
    for c in s:
        if c in morse:
            result += morse[c] + "/"
        elif c == " ":
            result += "/"
    return result

print(to_morse("HELLO WORLD"))
# ...././.-../.-../---//.--/---/.-./.-../-../
```

Slovník v Pythonu – příklady použití III

Substituční šifra

```
def encrypt(text, subst):
    result = ""
    for c in text:
        if c in subst:
            result += subst[c]
        else:
            result += c
    return result

my_cipher = {'A': 'Q', 'B': 'W', 'C': 'E'} # etc.

print(encrypt("BAC", my_cipher))
# WQE
```

Shrnutí

Datové typy

- abstraktní (rozhraní) vs. konkrétní (implementace)

Seznam

- posloupnost prvků; typicky umožňuje přidávat a odebírat
- v Pythonu: ntice (tuples) – něco jako neměnné seznamy

Zásobník/Fronta

- LIFO / FIFO

Množina

- udržuje neuspořádané jedinečné prvky
- umožňuje přidávat, odebírat, vyhledávat

Slovník

- množina záznamů (klíč, hodnota)
- umí všechno co množina a navíc indexovat klíčem