4. Pole

BAB37ZPR – Základy programování

Stanislav Vítek

Katedra radioelektroniky
Fakulta elektrotechnická
České vysoké učení v Praze

Přehled témat

- Část 1 – Pole

Část I

Pole

I. Pole

Datové typy v Pythonu

Jednoduché typy

- celé číslo, reálné číslo, logická hodnota

primitive data type

Složené typy / datové struktury

- řetězec, *n*-tice *(tuple)*, **pole**

composite/compound data type, data structures

- Hierarchicky sdružují data
- Související data jsou uložena a manipulována spolu
- Pro zvýšení efektivity programování i vykonávání
- Operace na datových strukturách
  - vytvoření
  - čtení a modifikace jednotlivých složek (elementů)
  - vyhledávání, přidávání, odebírání, ...

Pole

- Obsahuje `N` elementů (objektů, prvků), indexovaných od `0`
- Přímý přístup *(random access)*
  - Pro čtení i zápis, v konstantním čase
- Vytvoření

```
a=[0.3,0.6,0.1]
a
type(a)
```

- Čtení prvku

```
a[1]
a[0]
```

- Změna prvku

```
a[2]=1.5
a
```

Pole

- Obsahuje `N` elementů (objektů, prvků), indexovaných od `0`
- Přímý přístup *(random access)*
  - Pro čtení i zápis, v konstantním čase
- Vytvoření

```
a=[0.3,0.6,0.1]
a
type(a)
```

- Čtení prvku

```
a[1]
a[0]
```

- Změna prvku

```
a[2]=1.5
a
```

Pole

- Obsahuje `N` elementů (objektů, prvků), indexovaných od `0`
- Přímý přístup *(random access)*
  - Pro čtení i zápis, v konstantním čase
- Vytvoření

```
a=[0.3,0.6,0.1]
a
type(a)
```

- Čtení prvku

```
a[1]
a[0]
```

- Změna prvku

```
a[2]=1.5
a
```

Operace s polem

- Výpis pole

```
a=[0.3,0.6,0.1]
print(a)
```

```
  [0.3, 0.6, 0.1]
```

- Index může být výraz

```
s=0.
for i in range(3):
    s+=a[i]
    print("a[%d]=%f" % (i,a[i]))
print(s)
```

```
  a[0]=0.300000
  a[1]=0.600000
  a[2]=0.100000
  0.9999999999999999
```

Pole různých typů

```
a=[0.3,0.6,0.1]
print(a[0])
```

```
b=[3,1,4,1,5,9,2]
print(b[2])
```

```
barvy=["srdce","listy","kule","zaludy"]
print(barvy[3])
```

```
bits=[True,False]
print(bits)
```

Homogenní pole → všechny prvky jsou stejného typu.

Funkce a pole – unární operace

```
>>> a=[0.3,0.6,0.1]
>>> print(a)
[0.3,0.6,0.1]
>>> len(a)
3
>>> sum(a)
1
>>> max(a)
0.6
>>> bits=[True,False]
>>> all(bits)
False
>>> any(bits)
True
```

Funkce a pole – binární operace

- Spojování, opakování

```
>>> a=[0.3,0.6,0.1]
>>> b=[0.7,0.9]
>>> a+b
[0.3, 0.6, 0.1, 0.7, 0.9]
>>> b*3
[0.7, 0.9, 0.7, 0.9, 0.7, 0.9]
```

Vytvoření pole 1/3

- Výčtem

```
>>> a=[0.3,0.6,0.1]
```

- Opakováním prvků

```
>>> a=10*[0]
>>> a
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
```

Opakování používejte pouze pro primitivní nebo neměnné typy.

```
>>> a=3*[[1,2]]
>>> a
[[1, 2], [1, 2], [1, 2]]
>>> a[1][0]=10
>>> a
[[10, 2], [10, 2], [10, 2]]
```

Vytvoření pole 2/3

- Z posloupnosti

```
list(range(1,11))
```

```
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

V Pythonu se tento druh pole jmenuje `list`

- Přidáváním na konec

```
a=[]
for i in range(10):
    a+=[0.0]
print(a)
```

```
[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
```

Přidávání může být časově náročné.

Vytvoření pole 3/3

- Výrazem *(list comprehension)*

```
[i for i in range(1,11)]
```

```
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

```
[i*i for i in range(1,11)]
```

```
[1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
```

```
[0. for i in range(1,11)]
```

```
[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
```

Elegantní, ale specialita Pythonu — nemusíte si pamatovat.

Indexace

`x[i]`

- *i*-tý prvek pole `x`
- pro sekvenční typy: pole, řetězce, *n*-tice

```
a=[2,7,1]
print(a[2])
```

```
s="Ferda"
print(s[2])
```

```
t=(1,2)
print(t[0])
```

Indexace – záporné indexy

`x[i]=x[len(x)+i]`

Specialita Pythonu.

`a=[6,7,5,2,9]`

```
>>> a[-1]
9
>>> a[2]
5
>>> a[-2]
2
```

Řezy pole

```
x[i:j]=[x[i], x[i+1], ..., x[j-1]
x[i:]=x[i:len(x)]
x[:j]=x[0:j]
x[:]=x[0:len(x)]=x
```

Specialita Pythonu, podobný přístup např. v Matlabu

- Příklad:

```
a=[6,7,5,2,9]
print(a[2:4])
```

```
[5, 2]
```

```
print(a[:3])
```

```
[6, 7, 5]
```

Příklad: jména dnů v týdnu

Úkol: převeďte $i \in \{0, \ldots, 6\}$ na jméno dne.

```
def jmeno_dne(i):
  if i==0: return "pondeli"
  elif i==1: return "utery"
  elif i==2: return "streda"
  elif i==3: return "ctvrtek"
  elif i==4: return "patek"
  elif i==5: return "sobota"
  elif i==6: return "nedele"
  else: return "???"
print(jmeno_dne(3))
```

```
ctvrtek
```

Příklad: jména dnů v týdnu – pomocí pole

```
jmena_dni=["pondeli","utery","streda","ctvrtek",
           "patek","sobota","nedele"]
print(jmena_dni[3])
```

Uzávorkovaný výraz lze rozdělit na více řádek.

```
ctvrtek
```

```
def jmeno_dne(i):
    return jmena_dni[i]
print(jmeno_dne(3))
```

```
ctvrtek
```

Příklad: průměr a směrodatná odchylka

$$\mu_x = \frac{1}{N}\sum_{i=1}^{N} x_i, \qquad \sigma_x = \sqrt{\frac{1}{N-1}\sum_{i=1}^{N}(x_i - \mu_x)^2}$$

```
def mean(v):
  "Calculate a mean of a vector"
  s=0.
  for i in range(len(v)):
    s+=v[i]
  return(s/len(v))
```

Řetězec za hlavičkou funkce slouží k dokumentaci. Lze ho zobrazit příkazem `help(mean)`.

Příklad: průměr a směrodatná odchylka (2)

$$\mu_x = \frac{1}{N}\sum_{i=1}^{N} x_i, \qquad \sigma_x = \sqrt{\frac{1}{N-1}\sum_{i=1}^{N}(x_i - \mu_x)^2}$$

```
import math

def stdev(v):
  "Calculate a corrected sample standard deviation"
  m=mean(v)
  s=0.
  for i in range(len(v)):
    s+=(v[i]-m)**2
  return math.sqrt(s/(len(v)-1))
```

Příklad: průměr a směrodatná odchylka (3)

Vypočítáme μ, σ pro $x_1 = 0, \ldots, x_{1001} = 1000$

```
a=list(range(1001))
print("mean=", mean(a), " sigma=", stdev(a))
```

```
mean= 500.0  sigma= 289.10811126635656
```

Pro spojitou uniformní distribuci $[0, 1000]$ je

- $\mu = 500$ a
- $\sigma = \frac{1000}{\sqrt{12}} \approx 288.67$

Pole jako argument cyklu

- Místo

```
def mean(v):
  "Calculate a mean of a vector"
  s=0.
  for i in range(len(v)):
    s+=v[i]
  return(s/len(v))
```

- Můžeme psát

```
def mean(v):
  "Calculate a mean of a vector"
  s=0.
  for x in v:
    s+=x
  return(s/len(v))
```

Argumentem cyklu `for` je *sekvence*, například pole.

Pole jako argument cyklu (2)

- Výsledek je stejný

```
def mean(v):
  "Calculate a mean of a vector"
  s=0.
  for x in v:
    s+=x
  return(s/len(v))


a=list(range(1001))
print("mean=", mean(a))
```

```
mean=500
```

Použití funkcí pole

- Místo:

```
def mean(v):
  s=0.
  for x in v:
    s+=x
  return(s/len(v))
```

- Můžeme psát

```
def mean(v):
  "Calculate a mean of a vector"
  return(sum(v)/len(v))
```

Pokud můžete, používejte existující funkce.

Použití funkcí pole (2)

- Místo:

```
def stdev(v):
  "Calculate a corrected sample standard deviation"
  m=mean(v)
  s=0.
  for i in range(len(v)):
    s+=(v[i]-m)**2
  return math.sqrt(s/(len(v)-1))
```

- Můžeme psát

```
def stdev(v):
  "Calculate a corrected sample standard deviation"
  m=mean(v)
  s=sum([(x-m)**2 for x in v])
  return math.sqrt(s/(len(v)-1))
```

Použití funkcí pole (2)

- Funkci

```
def stdev(v):
  "Calculate a corrected sample standard deviation"
  m=mean(v)
  s=sum([(x-m)**2 for x in v])
  return math.sqrt(s/(len(v)-1))
```

- lze dále zkrátit

```
def stdev(v):
  return math.sqrt(sum([(x-mean(v))**2 for x in v])/
                       (len(v)-1))
a=list(range(1001))
print("mean=",mean(a)," sigma=",stdev(a))
```

```
mean= 500.0  sigma= 289.10811126635656
```

Výsledek se nezměnil. Zkracování ovšem může zhoršit srozumitelnost kódu.

Příklad: Náhodná permutace

Vytvořte náhodnou permutace čísel $0, 1, \ldots, N-1$

Myšlenka

- Uložíme do pole počáteční permutací $0, 1, \ldots, N-1$
- Budeme *vyměňovat* vždy dva prvky
- Aktuální prvek $i = 0, \ldots, N-2$ vyměníme s náhodně vybraným prvkem na pozici $j = i, i+1, \ldots, N-1$

Příklad: Náhodná permutace

Vytvořte náhodnou permutace čísel $0, 1, \ldots, N-1$

Myšlenka

- Uložíme do pole počáteční permutací $0, 1, \ldots, N-1$
- Budeme *vyměňovat* vždy dva prvky
- Aktuální prvek $i = 0, \ldots, N-2$ vyměníme s náhodně vybraným prvkem na pozici $j = i, i+1, \ldots, N-1$

Příklad: Náhodná permutace (2)

```
import random

def permutation(n):
    "Create a random permutation of integers 0..n-1"
    p=list(range(n))
    for i in range(n-1):
        r=random.randrange(i,n)
        temp=p[r]
        p[r]=p[i]
        p[i]=temp
    return(p)
```

Příklad: Náhodná permutace (3)

- Vyzkoušíme:

```
print(permutation(10))
```

```
[9, 5, 3, 2, 4, 8, 0, 6, 1, 7]
```

```
print(permutation(10))
```

```
[7, 0, 6, 4, 2, 9, 3, 8, 5, 1]
```

```
print(permutation(10))
```

```
[5, 8, 1, 7, 0, 9, 6, 4, 2, 3]
```

Kontrolní tisky

- Jak to vlastně funguje?

```
def permutation(n):
    "Create a random permutation of integers 0..n-1"
    p=list(range(n))
    print("p=",p)
    for i in range(n-1):
        r=random.randrange(i,n)
        temp=p[r]
        p[r]=p[i]
        p[i]=temp
        print("i=%d r=%d p=%s" % (i,r,str(p)))
    return(p)

permutation(5)
```

Kontrolní tisky (2)

```
p= [0, 1, 2, 3, 4]
i=0 r=0 p=[0, 1, 2, 3, 4]
i=1 r=4 p=[0, 4, 2, 3, 1]
i=2 r=3 p=[0, 4, 3, 2, 1]
i=3 r=4 p=[0, 4, 3, 1, 2]
```

Toto je velmi obecná a užitečná technika ověření funkčnosti programů.

Příklad: Permutace pole

- Vytiskneme prvky pole v náhodném pořadí:

```
barvy=["srdce","listy","kule","zaludy"]
p=permutation(len(barvy))
for i in range(len(barvy)):
    print(barvy[p[i]], end=" ")
```

```
zaludy kule listy srdce
```

Pole v novém pořadí:

```
print([ barvy[i] for i in p])
```

```
['zaludy', 'kule', 'listy', 'srdce']
```

Příklad: Permutace pole

- Vytiskneme prvky pole v náhodném pořadí:

```
barvy=["srdce","listy","kule","zaludy"]
p=permutation(len(barvy))
for i in range(len(barvy)):
    print(barvy[p[i]], end=" ")
```

```
zaludy kule listy srdce
```

Pole v novém pořadí:

```
print([ barvy[i] for i in p])
```

```
['zaludy', 'kule', 'listy', 'srdce']
```

Příklad: Házení dvěma kostkami

Jaká je pravděpodobnost, že padne součet s?

$$P(s) = \frac{\text{počet příznivých}}{\text{počet celkem}} = \frac{6 - |s - 7|}{6^2}$$

s	počet možnosti	P(s)
2	1	0.028
3	2	0.056
4	3	0.083
5	4	0.111
6	5	0.139
7	6	0.167
8	5	0.139
9	4	0.111
10	3	0.083
11	2	0.056
12	1	0.028

Příklad: Házení dvěma kostkami

Jaká je pravděpodobnost, že padne součet s?

$$P(s) = \frac{\text{počet příznivých}}{\text{počet celkem}} = \frac{6 - |s - 7|}{6^2}$$

s	počet možnosti	P(s)
2	1	0.028
3	2	0.056
4	3	0.083
5	4	0.111
6	5	0.139
7	6	0.167
8	5	0.139
9	4	0.111
10	3	0.083
11	2	0.056
12	1	0.028

Simulace házení dvěma kostkami

```
import random

h=[0]*13 # četnost výskytu součtu h[s]
n=100000

# Simulace n dvojic hodů
for i in range(n):
  x=random.randrange(1,7)
  y=random.randrange(1,7)
  s=x+y
  h[s]+=1
```

Simulace házení dvěma kostkami (2)

```
for s in range(2,13): # Tisk pravděpodobností
    anal=(6-abs(s-7))/36
    simul=h[s]/n
    print("s=%2d  P(s)=analyticky %0.3f    "
          "simulace %0.3f    chyba %6.3f" %
          (s,anal,simul,anal-simul))
```

```
s= 2  P(s)=analyticky 0.028  simulace 0.028  chyba -0.000
s= 3  P(s)=analyticky 0.056  simulace 0.056  chyba -0.000
s= 4  P(s)=analyticky 0.083  simulace 0.083  chyba  0.001
s= 5  P(s)=analyticky 0.111  simulace 0.113  chyba -0.001
s= 6  P(s)=analyticky 0.139  simulace 0.137  chyba  0.002
s= 7  P(s)=analyticky 0.167  simulace 0.168  chyba -0.001
s= 8  P(s)=analyticky 0.139  simulace 0.138  chyba  0.001
s= 9  P(s)=analyticky 0.111  simulace 0.113  chyba -0.002
s=10  P(s)=analyticky 0.083  simulace 0.083  chyba  0.001
s=11  P(s)=analyticky 0.056  simulace 0.054  chyba  0.001
s=12  P(s)=analyticky 0.028  simulace 0.028  chyba -0.000
```

I. Pole

Pole

Hodnoty a reference

Hodnotová semantika

- U objektu je důležitá hodnota, nikoliv identita. Proměnná reprezentuje hodnotu.
- Primitivní typy v Pythonu se chovají jako hodnoty *(values)*
- Přiřazení vytvoří nový objekt.

```
a=7
b=a
a=6
print(a)
```

```
6
```

```
print(b)
```

```
7
```

Hodnotová semantika

- U objektu je důležitá hodnota, nikoliv identita. Proměnná reprezentuje hodnotu.
- Primitivní typy v Pythonu se chovají jako hodnoty *(values)*
- Přiřazení vytvoří nový objekt.

```
a=7
b=a
a=6
print(a)
```

```
6
```

```
print(b)
```

```
7
```

Referenční semantika

- Proměnná typu pole je referencí/odkazem *(reference,link)*
- Přiřazení vytvoří nový *odkaz* na existující objekt.

```
a=[7,3]
b=a
a[1]=6
print(a)
```

```
[7, 6]
```

```
print(b)
```

```
[7, 6]
```

- Pole lze měnit *(mutable)*
- Sdílení odkazů *(sharing,aliasing)*

Referenční semantika

- Proměnná typu pole je referencí/odkazem *(reference,link)*
- Přiřazení vytvoří nový *odkaz* na existující objekt.

```
a=[7,3]
b=a
a[1]=6
print(a)
```

```
[7, 6]
```

```
print(b)
```

```
[7, 6]
```

- Pole lze měnit *(mutable)*
- Sdílení odkazů *(sharing,aliasing)*

Neměnnost

- Vlastnost datového typu.
- Neměnné objekty po vytvoření změnit nelze — řetězce, *n*-tice

```
s="Ahoj"; s[0]="a"
```

```
TypeError: 'str' object does not support item assignment
```

- Neměnné typy (řetězce, *n*-tice) se také chovají jako hodnoty

```
r=s
r="Nazdar"
print(a)
```

```
Nazdar
```

```
print(b)
```

```
Ahoj
```

Proměnné v Pythonu nejsou hodnoty, ale odkazy (references).

Neměnnost

- Vlastnost datového typu.
- Neměnné objekty po vytvoření změnit nelze — řetězce, *n*-tice

```
s="Ahoj"; s[0]="a"
```

```
TypeError: 'str' object does not support item assignment
```

- Neměnné typy (řetězce, *n*-tice) se také chovají jako hodnoty

```
r=s
r="Nazdar"
print(a)
```

```
Nazdar
```

```
print(b)
```

```
Ahoj
```

Proměnné v Pythonu nejsou hodnoty, ale **odkazy (references)**.

Práce s neměnnými objekty

- Jak lze s neměnnými objekty pracovat?
- Vytvoříme objekt nový, nezávislý na starém.

```
s="Ahoj"
r="a"+s[1:]
```

```
>>> r
'ahoj'
>>> s
'Ahoj'
```

Vedlejší efekty funkcí

- Funkce může změnit své změnitelné *(mutable)* parametry

```
def add_one(x):
    for i in range(len(x)):
        x[i]+=1

v=[1,2,3]
add_one(v)
print(v)
```

```
[2, 3, 4]
```

- Funkce není čistá *(impure)*.
- Vedlejším efektům se pokud možno vyhněte.

Vedlejší efekty funkcí

- Funkce může změnit své změnitelné *(mutable)* parametry

```
def add_one(x):
    for i in range(len(x)):
        x[i]+=1

v=[1,2,3]
add_one(v)
print(v)
```

```
[2, 3, 4]
```

- Funkce není čistá *(impure)*.
- Vedlejším efektům se pokud možno vyhněte.

Nahrazení vedlejších efektů

```
def add_one_clean(x):
    return [x[i]+1 for i in range(len(x))]
v=[1,2,3]
v=add_one_clean(v)
print(v)
```

```
[1, 2, 2]
```

Kopírování polí

- Přiřazení proměnné typu pole vytvoří nový odkaz na stejné pole

```
a=[7,3]
b=a
a[1]=6
print("a=",a, "b=",b)
```

```
a= [7, 6] b= [7, 6]
```

- Kopírováním se vytvoří nový objekt se stejným obsahem

```
a=[7,3]
b=a.copy() # Lze psát též b=a[:]
a[1]=6
print("a=",a, "b=",b)
```

```
a= [7, 6] b= [7, 3]
```

Kopírování polí

- Přiřazení proměnné typu pole vytvoří nový odkaz na stejné pole

```
a=[7,3]
b=a
a[1]=6
print("a=",a, "b=",b)
```

```
a= [7, 6] b= [7, 6]
```

- Kopírováním se vytvoří nový objekt se stejným obsahem

```
a=[7,3]
b=a.copy() # Lze psát též b=a[:]
a[1]=6
print("a=",a, "b=",b)
```

```
a= [7, 6] b= [7, 3]
```

Neměnost

Výhody

- Vyloučení vedlejších efektů
- Méně chyb
  - Kdy kopírovat, co lze přepsat
  - Vzdálený kód mění proměnné
- Snazší optimalizace
- Snazší paralelizace

Nevýhody

- Trochu menší expresivita.
- Objektů vzniká velké množství, alokace/dealokace paměti.
- Nutnost kopírování.
- Paměťová a výpočetní náročnost.

Existují techniky jak kopírování omezit.

Neměnost

Výhody

- Vyloučení vedlejších efektů
- Méně chyb
  - Kdy kopírovat, co lze přepsat
  - Vzdálený kód mění proměnné
- Snazší optimalizace
- Snazší paralelizace

Nevýhody

- Trochu menší expresivita.
- Objektů vzniká velké množství, alokace/dealokace paměti.
- Nutnost kopírování.
- Paměťová a výpočetní náročnost.

Existují techniky jak kopírování omezit.

Neměnost

Výhody

- Vyloučení vedlejších efektů
- Méně chyb
  - Kdy kopírovat, co lze přepsat
  - Vzdálený kód mění proměnné
- Snazší optimalizace
- Snazší paralelizace

Nevýhody

- Trochu menší expresivita.
- Objektů vzniká velké množství, alokace/dealokace paměti.
- Nutnost kopírování.
- Paměťová a výpočetní náročnost.

Existují techniky jak kopírování omezit.

Dvourozměrné matice

- Pole polí

```
a=[[1,0,2,3],[0,2,3,1],[3,0,2,5]]
print(a)
```

```
[[1, 0, 2, 3], [0, 2, 3, 1], [3, 0, 2, 5]]
```

```
print(len(a))
```

```
3
```

```
print(a[1])
```

```
[0, 2, 3, 1]
```

```
print(a[1][2])
```

```
3
```

Tisk matice

```
def print_2d_matrix(a):
    for i in range(len(a)):
        print(a[i])

print_2d_matrix(a)
```

```
[1, 0, 2, 3]
[0, 2, 3, 1]
[3, 0, 2, 5]
```

Závěr

- Pole
  - často používaná datová struktura
  - obsahuje n prvků (nejčastěji stejného typu)
  - k prvkům přistupujeme pomocí celočíselného indexu
  - prvky pole mohou být i složené datové typy
- Proměnné jsou reference, aliasing
- Neměnné (immutable) typy, hodnotová semantika
- Příklady použití polí