JEŠTĚ K FUNKCÍM...

Funkce jsou objekty

takže se dají ukládat do proměnných...

```
tisk = print
tisk(f"{2}*{5}={2*5}")
```

Funkce CHYBA

...i předávat jako parametry

```
def tabulka(f):
    for i in range(10):
        print(f"{i:2d}:{f(i):+.2f}")
        # :+.2f = znamenko, 2 des. Mista

import math
tabulka( math.sin )
tabulka( math.cos )
```

Lambda funkce

lambda calculus, Alonzo Church, 1930

```
tabulka( lambda x: 1/(x+1) ) # 1/(x+1), NE 1/x!
tabulka( lambda x: x*x )

pricti1 = lambda x:x+1
pricti1(27)
   28

soucet = lambda x,y: x+y
soucet(12,25)
   37
```

Lambda funkce [2]

je to jediný výraz
nemůžou obsahovat příkazy

známy též jako: anonymní funkce nebo lambda-výrazy

použití: funkce vyžadované jako parametr:

```
lidi = [ ['Bořík','Weber'],['Anna','Zelená'],
         ['Cyril','Sláma'],['David','Hroch']]

sorted( lidi )
sorted( lidi, key = lambda x: x[1]+x[0] )
```

Closure

funkce může jako svůj výsledek vrátit funkci...

...která si zachová přístup k proměnným nadřízené funkce

```
def pricitacka( kolik ):
    def f(x):
        return x+kolik
    return f

p1 = pricitacka(1)
p1(10)
  11
```

k čemu to je?

vyhnout se používání globálních proměnných

ukrývání dat (lépe než je ukrývají objekty)

Closure [2]

```
def soucet():
    N = 0
    def secti(x,y):
        nonlocal N
        N += 1
        if N>3:
            print("Už sčítám moc dlouho!")
            return -1
        return x+y
    return secti
```

MODULY 3

NumPy

balíček pro vědecké výpočty

- vícerozměrná pole
- nástroje pro zapojení kódu v C/C++ a Fortranu
- funkce pro lineární algebru, Fourierrovu transformaci a náhodná čísla

Open source

Od roku 2006

Nezisková organizace

NumFOCUS

NumPy [2]

vytvoření a naplnění pole:

```
import numpy as np
a = np.array([1,2,3,4,5,6,7])
```

prvky pole jsou stejného typu! :

```
print( a.dtype )
a[2] = 12.75  # převede na int32
b = np.array( [ [1,2], [3,4] ], dtype=complex )
```

změna typu pole:

```
af = a.astype('float64')
```

NumPy [3]

vytvoření pole s hodnotami:

```
nuly = np.zeros( (3,4) )
jednicky = np.ones( (2,3,4) )
neinicializovane = np.empty( (10,10) )

c = np.arange(20)
d = np.arange( 0, 2, 0.3 )

x = np.linspace( 0, 2*numpy.pi, 100 )
y = np.sin(x) # pro každý prvek z x
              #    vytvoří prvek v y
r = np.random.random(100) # náhodná z [0;1)
```

NumPy [4]

změna tvaru pole:

```
aa = a.reshape(4,5)
```

tisk:

- zarovnání
- i 3D pole
- „…“, pokud je příliš veliké:
  ```
  print(np.arange(10000).reshape(100,100))
  ```

NumPy [5]

operace:

```
a2 = a+a
a3 = 10*a
aaa = a**2
mala = a<50
sb = np.sqrt(b)
```

maticové operace:

```
A = np.arange(16).reshape(4,4)
AA = A@A

N = 1000*1000
x = np.random.random(N)
y = np.random.random(N)
sum(x*x+y*y<1)/N*4
```

NumPy [6]

dosazení:

- `b = a`

  dosazení nic nekopíruje! (jen ukazatel)
- `c = a.view()` `# jiný pohled na stejná data`
- `d = a[5:]` `# jiný pohled na stejná data`
- `d = a.copy()` `# tohle vytvoří kopii`

indexy:

```
a = 10*np.arange(10)
i = np.array( [2,1,3,2,7] )
a[i]
array([20, 10, 30, 20, 70])
```

NumPy [7]

Matplotlib & NumPy

HRY
(KONEČNÉ, S ÚPLNOU INFORMACÍ)
(známe z Algoritmizace, chceme naprogramovat)

Konečné hry s úplnou informací

hra dvou hráčů
konečná....skončí po konečném počtu kroků
s úplnou informací....oba hráči mají
všechny informace o stavu hry

Příklad: odebírání zápalek, piškvorky na konečné ploše, dáma, šachy, go...

Vyhrávající a prohrávající pozice

(kdyby neexistovala remíza)

- koncová pozice, ve které hráč na tahu prohrál,
  je prohrávající
- koncová pozice, ve které hráč na tahu vyhrál,
  je vyhrávající

- pozice, ze které lze hráče dostat do prohrávající pozice,
  je vyhrávající
- pozice, ze které nelze hráče dostat do prohrávající pozice,
  je prohrávající

=> (kdyby neexistovala remíza)
  každá dostupná pozice je buď vyhrávající nebo prohrávající

Strom hry

(nemusí být strom)

= Graf, vrcholy – stavy hry, hrany – možné tahy

Pokud existuje remíza, jsou kromě prohrávajících a vyhrávajících pozic ještě neprohrávající pozice.

Věta:

U konečné hry s úplnou informací alespoň pro jednoho hráče existuje neprohrávající strategie.

příklad: zápalky 1..3

příklad: zápalky z více hromádek

Hry s ohodnocením

= Každá cílová pozice je ohodnocena číslem
(namísto vyhrál-prohrál).
Jeden hráč se snaží dosáhnout maximálního výsledku,
druhý minimálního.

Hra s nulovým součtem

= Hra, ve které zisk jednoho hráče
je roven ztrátě druhého hráče.

Algoritmus MINIMAX

Strom hry se ohodnocuje od listů – koncových pozic,
(tam už víme, jak hra dopadla a ohodnocení známe).
Hodnota vrcholu-stavu se spočte jako
maximum/minimum hodnot synů,
podle toho,
zda je na tahu
maximalizující nebo minimalizující hráč.

příklad: řada čísel – součet dolů vs. součet nahoru

příklad: tabulka čísel – součet dolů vs. součet nahoru

Negamax

= Alternativa MINIMAXu,
namísto střídání MIN a MAX se počítá -MAX(-...).
Snazší naprogramování.

```
H := max( min( max( min(...))))
```

vs.

```
H := max( - max( - max( - max(-...))))
```

Heuristika

= rada, něco, co „obvykle dává dobré výsledky“

V užším smyslu (nic nezkazí)

pořadí prozkoumávání tahů...

V širším smyslu (náhrada úplného řešení).

příklad s loupežníky...