Pole

podruhé

- mají plno metod, které ovšem nevíme, co jsou (a tudíž se jim zkusíme přiměřeně vyhýbat).
- Jak udělat pole délky k plné nul?
- `pole=[0]*k`
- Ke stringu lze přistoupit jako k poli znaků (readonly):
- `a='nazdar'`
- `a[1]=='a'`
- `a[1]='e'` neprojde.
- Zjištění délky: `len(pole)`

for-cyklus

- Chceme iterovat přes posloupnost (hodnoty pole, znaky stringu,...):
- ```
  for i in pole:
      telo_cyklu
      jako_u_ifu
      ci_whilu
  ```
- Chceme-li iterovat k krát, zavoláme funkci `range`:
- `for i in range(k)`
- vrací objekt (nikoliv samotný seznam),
- může brát až tři parametry: `range(start,stop,step)`
  ```

Hodnotové a referenční typy

jsou záludné a říká se jim *immutable* a *mutable*

Co udělá tento kód?

```
a=10
b=a
a=15
print(b)
```

A co tento?

```
a=[10]
b=a
a[0]=15
print(b[0])
```

Integer je hodnotový typ, pole referenční.

Funkce a procedury

nám umožní shrnout kus kódu na hromádku a volat odkudkoliv

- Známe funkce `print`, `input`, `int` a `str`, tedy funkce umíme volat:
- `jmeno(par1,par2,par3,... jmeno=hodnota,...)`
- Definujeme: `def jmeno(parametry,... impl_par=hod,...):`
- Tak zahájíme blok, do kterého napíšeme tělo.
- Ve funkci používáme implicitně lokální proměnné,
- chceme-li proměnnou globální, řekneme: `global jmeno`.
- Návratová hodnota `return vyraz;`

Příklad

definice funkce

```
def secti(a,b):
    c=a+b
    return c
print(secti(5,3))
```

Předávání parametrů

se děje přiřazením

- Hodnotové (immutable) se okopírují,
- referenční (mutable) se předají referencí.

Návratová hodnota

může být tuple

Příklad:

```
def zvyso1(a,b):
    return a+1,b+1
a,b=zvyso1(a,b)
```

Podobně lze i přiřazovat:

```
a,b,c=1,2,3
```

Což opět poutá naši pozornost k nežádoucím detailům.

Implicitní parametry

jsou alternativou přetěžování, které uvidíme v létě

- V Pascalu či C je funkce určena názvem,
- v C++, C#, Javě,... je funkce určena názvem a strukturou parametrů (tzv. přetěžování),
- v Pythonu nelze dvakrát definovat touž funkci,
- ale lze říci, že parametry, které nejsou nastavené, mají implicitní hodnotu (přetěžování se beztak často používá tímto způsobem).

Implicitní parametry

příklad

```
def vynasob(a,b,c=1,d=1,e=1):
    return a*b*c*d*e
    #Parada, bude to fungovat pro 2 - 5 argumentu!
print(vynasob(2019,10,8))
```

Neznámý počet parametrů se řeší hvězdičkou a případně vzniká tuple:

```
def vynasobvse(*a):
    soucin=1
    for i in a:
        soucin*=i
    return soucin
```

A nyní oboje dohromady

a vyjde nám funkce `print`

- `def print(*objects, sep=' ', end='\n', file=sys.stdout,flush=False):...`
- Takto vypadá hlavička funkce `print`, kterou používáme. Co to znamená?
- Funkce umí vytisknout neomezený počet parametrů,
- umí nastavit i různé dekorace (oddělovače, výstupní soubor),
- a tyto dekorace musíme nastavit explicitním předáním implicitního parametru, protože nepojmenované parametry stáhne proměnná `objects`.

Vnořené funkce

jsou viditelné jen z funkce, kde jsou definované

```
def f():
    x=1 #lokální proměnná
    def g(): #vnořená funkce viditelná z f
        print(x)
    g() # toto zafunguje
f()# toto také, ale volání
# g() zde by spadlo.
```

Vnořené funkce

a klíčové slovo `nonlocal`

```
def f():
    x=1 # stale lokální proměnná
    def g():
        nonlocal x # promenna x z funkce f
        print(x)
        x=5
    g()
    print(x)# a v x je petka
```