Cykly

- `while` podminka `:`
  prikazy_odindentovane
  Opakuj, dokud je `podminka` splněna.
- Lze použít i `else` jako u `if`,
- význam: Do `else`-větve vstoupíme, nebyla-li podmínka splněna již při první iteraci.

Příklad:

```
a=int(input('Napis cislo: '))
while a > 0 :
        if a % 2 == 1 :
                print(1)
        else:
                print(0)
        a=a // 2
```

Příklad vylepšený

Při programování je hlavní **myslet, jinak si** často zbytečně **přidělám práci!**

```
a=int(input('Napis cislo: '))
while a > 0 :
        print(a%2)
        a=a // 2
```

Příklad, hledání prvočíselného rozkladu:

```
i=2
a=int(input('Zadej cislo: '))
while i <= a :
    if (a // i)*i == a :
        print(i)
        a=a // i
    else:
        i=i+1
```

Příklad, hledání prvočíselného rozkladu vylepšený:

```
i=2
opakujeme=False
a=int(input())
while i <= a :
     if (a // i)*i == a :
          if opakujeme :
                print('*',end='')
          else:
             opakujeme=True
          print(i,end='')
          a=a // i
     else:
          i=i+1
```

Neotřelé předávání parametrů

opět odpoutává naši pozornost od důležitějších věcí

- `print('Nazdar')` – funkce `print` vypíše zadané parametry,
- `print('Nazdar',end='')` – vypíšeme a neodřádkujeme
- `print(*objects, sep=' ', end='\n', file=sys.stdout,flush=False)` – skutečná hlavička `print`.
- Funkce `print` před námi tedy schovává čtyři implicitní parametry, které mohou ovládat její fungování...
- ... čímž nevhodně poutá naši pozornost k detailům.

Pole či seznamy

umožňují uskladnit větší množství údajů

- Definujeme pomocí hranatých závorek, do kterých dáme prvky:
- `seznam=[1,2,3,4,5]`
- k prvkům přistupujeme také operátorem hranatých závorek:
- `print(seznam[2])`
- Chceme-li více po sobě jdoucích prvků:
- `seznam[odkud:kam]` – tzv. řezy,
- a nechceme-li prvky po sobě jdoucí, ale ob jeden:
- `seznam[odkud:kam:po_kolika]` (alias slice)
- Opět je před námi schován systém implicitních parametrů `[0:-1:1]`.
- Seznam tedy lze obrátit takto: `obraceny=seznam[::-1]` (ale na Algoritmizaci to takto nedělejte).

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))
```