Základy psaní programů

Shnuli jsme základní příkazy obou jazyků, které používáme:

Nejprve to je přiřazovací příkaz a volání procedury:

```
c := (a+b)/2 ;
writeln( c ) ;
```

```
c = (a+b)/2
print( c )
```

Následuje podíněný příkaz

```
if c<0 then c := 0;
```

```
if c<0:
  c = 0
```

nebo, když je příkazů více, musíme použít v Pascalu složený příkaz

```
if a>b then begin
  c := a;
  a := b;
  b := c;
end;
```

```
if a>b:
  c = a
  a = b
  b = c
```

Je potřeba zmínit, že pro prohazování obsahu dvou proměnných je správné v Pythonu použít *tuple* ('tici)

```
if a>b then begin
  c := a;
  a := b;
  b := c;
end;
```

```
if a>b:
  (a,b) = (b,a)
```

Ještě máme větev `else`

```
if a>b then
  writeln('a>b')
else
  writeln('a<=b');
```

```
if a>b:
  print('a>b')
else:
  print('a<=b')
```

Kvůli hromadění odsazování museli v Pythonu zavést `elif`

```
if a>b then
  writeln('a>b')
else if a=b then
  writeln('a=b')
else
  writeln('a<b');
```

```
if a>b:
  print('a>b')
elif a=b:
  print('a=b')
else:
  print('a<b')
```

Dalším strukturovaným příkazem je cyklus `while`

```
s := 0
a := 1
while a<=10 do begin
   s := s+a
   a := a+1
end;
```

```
s = 0
a = 1
while a<=10:
   s = s+a
   a = a+1
```

Cyklus `repeat` se rozhoduje o ukončení až na koci těla cyklu a v Pythonu musíme užít `break` na konci cyklu `while` (kód počítá celou část odocniny z x jen s použitím operací sčítání a porovnání)

```
x := 65
y := 1
i := 0
repeat
   i := i+1
   y := y+i+i+1
until y>x;

writeln(i);
```

```
x = 64
y = 1
i = 0
while True:
   i = i+1
   y = y+i+i+1
   if y>x:
     break

print(i)
```

Již víme, jak užitečný je příkaz for. Zde se ale liší koncepce obou jazyků – Pascal poavžuje za důležité tzv ordinální typy, zejména celá čísla a příkaz for je určen pro jejich procházení. Python pak považuje za důležité seznamy a jejich procházení. Proto:

```
for i := 0 to 10 do
  s := s+i*i
```

```
for i in range(10+1):
  s = s+i*i
```

Zde se zatím tváříme, že `range(start, stop, step)` zjednodušeně i `range(start, stop)` a `range(stop)`

vrací seznam čísel, tedy `range(5)` → `[0,1,2,3,4]`, `range(2,5)` → `[2,3,4]` a `range(20,5,-5)` → `[20,15,10]`.

Obrácené procházení, tedy $i = 10, 9, 8, .., 1$ dosáhmene takto:

```
for i := 10 downto 1 do
  s := s+i*i
```

```
for i in range(10,1-1,-1):
  s = s+i*i
```

Konečně ještě porovnáme, jak se deklarují funkce a jak jsou začleněny v programu.

```
program radaPi;

function soucetPi4(n:integer):real;
var k:integer;
    soucet : real;
    m1nk: integer;
begin
  soucet := 0;
  m1nk   := 1;

  for k:=0 to n do begin
     soucet := soucet + m1nk/(2*k+1);
     m1nk := -m1nk;
  end;

  soucetPi4 := soucet;
end;


begin
  Writeln( 4*soucetPi4(1000) );
end.
```

```
def soucetPi4(n):

# lokalni promenne
# neni potreba deklarovat dopredu

  soucet = 0
  m1nk   = 1

  for k in range(0,n+1):
    soucet = soucet + m1nk/(2*k+1)
    m1nk   = -m1nk

  return soucet


print(  4*soucetPi4(100) )
```

Výrazy konstrujeme pomocí operací `+ - * /`. Dělení celých čísel dá v obou případech reálný výsledek. (Pozor na Python verze 2!)

```
  s := a*b+c/d;
```

```
  s = a*b+c/d
```

Celočíslené výrazy často obsahují dělení se zbytkem:

```
  radek   := poradi div pocet;
  sloupec := poradi mod sloupec;
```

```
  radek   = poradi // pocet;
  sloupec = poradi % sloupec;
```

Složitější logické výrazy vyžadují pozornost:

```
if (radek=sloupec) or (radek=pocet-sloupec+1)
    and (sloupec and 1 <> 0) then ...
```

```
if radek==sloupec or \
   radek==pocet+1-sloupec and sloupec & 1:
```

Zde je pro povšimnutí použita též binární operace `and` resp. `&` testující sudost/lichost řádku a v Pythonu i fakt, že nenulový výraz se automaticky konvertuje na `True` a nula na `False`. Konečně, v Pythonu znamenají konce řádků i konec příkazu, proto je potřeba zpětným lomítkem označit, že řádek pokračuje.

V Pythonu máme navíc mocnění, ale zase složiěji píšeme i nejzákladnější funkce

```
  gamma := 1/sqrt(1-sqr(v/c))
```

```
  gamma = 1/math.sqrt(1-(v/c)**2)
```

Záměr použít nějakou knihovnu/modul musíme dát v obou jazycích vědět (obvykle) na začátku programu

```
uses math;
  ...
writeln( arccos(0.25) );
```

```
import math

print( math.acos( 0.25 ) )
```

Seznam a názvy jednotlivých funkcí najdete na `https://www.freepascal.org/docs-html/rtl/math/index-5.html` resp. `https://docs.python.org/3/library/math.html`