Príkaz priradenia

Príkaz priradenia slúži na priradenie hodnoty premennej. Má tvar **premenná = výraz**, kde premenná je identifikátor, znak „=“ sa číta „priraď“ a vyhodnotením výrazu sa získa hodnota určitého typu, ktorá sa priradí ako nová hodnota premennej na ľavej strane príkazu priradenia. Výrazom môže byť aj n-tica alebo zoznam.
Pokiaľ sa príkazom priradenia aktualizuje hodnota premennej na ľavej strane príkazu priradenia schémou premena = operácia s premennou, možno použiť skrátený zápis, napríklad

„klasický“ zápis	skrátený zápis	pôvodná hodnota	nová hodnota	
pocet = pocet + 1	pocet += 1	pocet: 5	6	
sucet = sucet + cislo	sucet += cislo	sucet: 10 cislo:12.4	22.4	
text = text + slovo	text += slovo	text: „Prvé“ slovo: „Druhé“	„PrvéDruhé“	
cislo = cislo // 2	cislo //= 2	cislo: 5	2	(5 div 2)
cislo = cislo % 2	cislo %= 2	cislo: 5	1	(5 mod 2)
cislo = cislo**2	cislo **= 2	cislo: 5	25	(5^2)

Dovolené je viacnásobné priradenie prem1 = prem2 = ... premN = hodnota. Napríklad sucet = pocet = 0.
Dovolený je aj tvar prem1, prem2,..., premN = výraz1, výraz2,..., výrazN. Vo výrazoch sa dosadia hodnoty ktoré mali premenné pred začiatkom vykonávania príkazu priradenia! Ak niektorá premenná nemala priradenú hodnotu (nebola ešte použitá) nastane výnimka NameError.
Tento tvar dovoluje elegantne naprogramovať výmenu dvoch hodnôt, napríklad premenných a a b:

```
>>> a,b = 5,7
>>> print("a",a," b",b)
a 5  b 7
>>> a,b = b,a
>>> print("a",a," b",b)
a 7  b 5
```

Štandardný vstup a výstupu

Vstupné hodnoty možno zadať príkazom priradenia pred spustením programu. Zadanie hodnoty užívateľom počas behu programu umožňuje funkcia **input**, ktorá má tvar: input("popisný_reťazec"). Zobrazí sa popisný reťazec a po zadaní hodnoty a stlačení klávesu Enter sa zadaná hodnota vráti do programu ako reťazec. Ak sa má hodnota priradiť ako číslo, treba vstupný reťazec konvertovať na požadovaný číselný typ (int alebo float).
Príklad použitia:

```
>>> meno = input("Ako sa voláš? ")
Ako sa voláš? Peter
>>> pocet_cisel = int(input("Zadaj počet čísel: "))
Zadaj počet čísel: 13
>>> moje_pi = float(input("Hodnota pi = "))
Hodnota pi = 3.14159
>>> print(meno, pocet_cisel, moje_pi)
Peter 13 3.14159
```

Jeden z variantov príkazu priradenia nám umožňuje zadať za sebou aj viacej vstupných hodnôt, ako napríklad

```
>>> r1, r2 = input("R1 v \u2126: "), input("R2 v \u2126: ")
R1 v Ω: 10
R2 v Ω: 20.4
```

Cez funkciu input možno vložiť aj viacej hodnôt oddelených medzerou, musíme si však uvedomiť, že budú importované ako reťazec a k jednotlivým hodnotám (ako reťazcom!) sa vieme dostať napríklad cez funkciu split, ktorá vytvorí zoznam reťazcov, ako to vidieť v ukážke:

```
>>> t = input("Zadaj teplotu ráno, na obed a večer ").split()
Zadaj teplotu ráno, na obed a večer 12.6 17.9 15.4
>>> print(t)
['12.6', '17.9', '15.4']
>>> tdna = ((float(t[0])+float(t[1])+2*float(t[2]))/4)
>>> print("Priemerná teplota dňa:",tdna)
Priemerná teplota dňa: 15.325
```

Na vypísanie hodnôt slúži príkaz **print**, ktorý má tvar: print(hodnota1, hodnota2,..., hodnotaN, sep="sep_reťazec", end="end_reťazec"), kde sep_reťazec je oddeľovací reťazec medzi jednotlivými vypisovanými hodnotami, štandardne je nastavená medzera (preto pri výpise viacerých hodnôt sú automaticky oddelené medzerou), a end_reťazec je reťazec na konci výpisu, štandardne znak \n. V ukážke sme oba parametre potlačili.

```
print("Namerané:", t, "priemerná teplota dňa:", tdna, sep="", end="")
print(" OK")
```

```
Namerané:['12.6', '17.9', '15.4']priemerná teplota dňa:15.325 OK
```

Parametre oddeľovací reťazec (sep=...) a reťazec na konci výpisu (end=...) možno použiť v príkaze print aj v kombinácii s formátovaním reťazca, t.j. print(reťazec.format(parametre), sep=..., end=...)! (Pozri študijný text formátovanie reťazcov.)

V prípade, že sa vo výstupnom reťazci vyskytujú znaky \n, Python ich interpretuje ako prechod na nový riadok. Potlačiť riadiace znaky možno zadaním r pred výstupný reťazec – ukážka:

```
>>> print("C:\Python\nove")
C:\Python
ove
>>> print(r"C:\Python\nove")
C:\Python\nove
```

Riadenie vykonávania programu

Určite si uvedomujete, že len so sekvenciou príkazov, ktoré sa všetky vykonajú za sebou tak, ako sú zapísané, v programovaní nevystačíme. Preto existujú príkazy vetvenia a cyklu, ktoré menia priamočiare vykonávanie príkazov v programe. Najprv si však musíme niečo ujasniť.

Pri použití riadiacich konštrukcií musí byť jednoznačne dané, ktoré príkazy sa majú opakovať (patria do cyklu) a ktoré už nie, alebo ktoré príkazy sa majú vykonať, ak je splnená podmienka a ktoré, ak podmienka nie je splnená (pri podmienenom príkaze). Takto k sebe patriacim príkazom budeme hovoriť, že patria do bloku. Rôzne programovacie jazyky riešia „zviditeľnenie“ blokov rôzne. V Delphi sa príkazy patriace do bloku uzatvárajú medzi begin a end, v jazyku C a Java sa uzatvárajú medzi zložené zátvorky {}. Python to rieši odsadením doprava príkazov patriacich do bloku (štandardne o štyri medzery). Keď odsadenie skončí, končí aj blok. Tiež musí byť zrejmé, kde končí podmienka a začína príkaz. To dáva najavo Python dvojbodkou.

Teda zápis

```
ak podmienka:
   príkaz1
   príkaz2
   ...
príkaz
...
```

znamená, že príkazy príkaz1, príkaz2,... patria do bloku a budú vykonané, len ak bude splnená podmienka. Príkaz príkaz bude vykonaný po príkaze ak (vždy). Ak by v uvedenom príkaze ak sa mal pri splnení podmienky vykonať len jeden príkaz, môžeme použiť aj zápis

```
ak podmienka: príkaz1
príkaz
```

keďže dvojbodka jednoznačne určuje, kde končí podmienka a začína príkaz1.

Podobne

```
ak podmienka:
   blok_príkazov_1
inak:
   blok_príkazov_2
príkaz
...
```

alebo

```
pokiaľ podmienka:
   opakujúci_sa_blok_príkazov
príkaz
...
```

Podmienený príkaz if

má viacej tvarov, postupne sa s nimi zoznámime.

Neúplné binárne vetvenie

- použijeme, ak sa príkazy majú vykonať, len ak je splnená určitá podmienka
- má tvar: if výraz_typu_boolean:
  blok_príkazov
- vykonanie: ak je podmienka splnená (výraz typu boolean nadobudol hodnotu True) vykoná sa blok_príkazov (ak podmienka nie je splnená, príkaz if je bez účinku)
- napríklad if cislo%2 == 0: print(cislo, "je párne"), if vek >= 18: print("Si zodpovedný pred zákonom")
- ak blok príkazov predstavuje len jeden príkaz, celý príkaz if môže byť zapísaný v jednom riadku (neodporúčame)

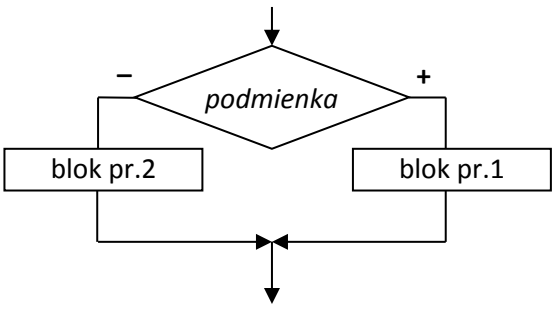
Úplné binárne vetvenie

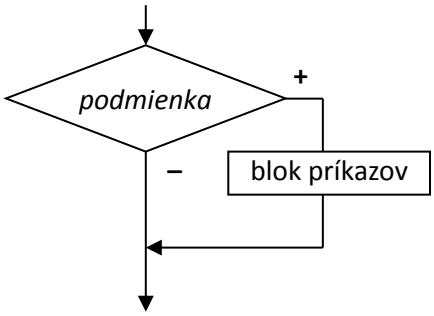
- použijeme, ak sa nejaké príkazy majú vykonať, len ak je splnená určitá podmienka a iné príkazy, ak podmienka nie je splnená
- má tvar: if výraz_typu_boolean:
  blok_príkazov_1
  else:
  blok_príkazov_2
- vykonanie: ak je podmienka splnená (výraz typu boolean nadobudol hodnotu True) vykoná sa blok_príkazov_1, ak podmienka nie je splnené (výraz nadobudol hodnotu False) vykoná sa blok_príkazov_2
- napríklad

```
cislo = int(input("Zadaj celé číslo: "))
if cislo%2 == 0:
   print(cislo, "je párne")
else:
   print(cislo, "je nepárne")
```

```
pH = float(input("Zadaj pH: "))
if 0<=pH<=14:
   print("Dovolená hodnota pH")
else:
   print("Nedovolená hodnota pH")
```

Grafické vyjadrenie binárneho vetvenia („+“ znamená, že je podmienka splnená, „-“ nesplnená)





N-árne vetvenie

Vráťme sa ku chémii, k pH. Aj pre dovolenú hodnotu pH môžu nastať tri situácie: pH menšie ako 7 - roztok je kyslý; pH rovné 7 - roztok je neutrálny a pH väčšie ako 7 - roztok je zásaditý.

Algoritmicky:

```
ak pH<7
tak píš("kyslý")              # podmienka pH<7 splnená
inak ak pH=7                  # podmienka pH<7 nesplnená, ešte zostávajú dve možnosti
     tak píš("neutrálny")     # podmienka pH<7 nesplnená, podmienka pH=7 splnená
     inak píš("zásaditý")     # podmienka pH<7 nesplnená, podmienka pH=7 nesplnená
```

v programe:

```
if pH<7:
   print("kyslý")
else:
   if pH==7:
      print("neutrálny")
   else:
      print("zásaditý")
```

Zápis a orientácia v zápise sa komplikujú vkladaním ďalších a ďalších vnorených príkazov if, ako to vidieť napríklad pri priradení bodov podľa tabuľky nižšie (slovo „viac“ znamená pod hornú hranicu danú hodnotou o riadok vyššie):

Úspešnosť	Body
97,5% - 100%	20
95,0% a viac	19
92,5% a viac	18
90,0% a viac	17
87,5% a viac	16
85,0% a viac	15
82,5% a viac	14
80,0% a viac	13
...	...
50,0% a viac	1
49,9% a menej	0

```
if uspesnost > 97.5:
   body = 20
else:
   if uspesnost > 95.0:
      body = 19
   else:
      if uspesnost > 92.5:
         body = 18
      else:
         if uspesnost > 90.0:
            body = 17
         ...
```

Python má pre takéto situácie nasledujúci tvar príkazu if

```
if výraz1:
   blok_príkazov_1
elif výraz2:
   blok_príkazov_2
...
elif výrazN:
   blok_príkazov_N
else:
   blok_príkazov
```

ktorého vykonanie je nasledovné: vyhodnotí sa výraz1, ak nadobudol hodnotu True, vykoná sa blok_príkazov_1 a príkaz if sa ukončí; ak nadobudol hodnotu False, vyhodnotí sa výraz2, ak nadobudol hodnotu True, vykoná sa blok_príkazov_2 a príkaz if sa ukončí; ak nadobudol hodnotu False, pokračuje sa vyhodnotením nasledujúceho výrazu atď. Ak ani jeden z výrazov nenadobudol hodnotu True, vykoná sa blok_príkazov za else. Ak vetva else chýba (je nepovinná) a všetky výrazy nadobudli hodnoty False, príkaz if je bez účinku (nevykoná sa nič).

Už na takej jednoduchej úlohe, ako je výpis pre roztok po zadaní pH sa dá ukázať, že správnych riešení je niekoľko (uvádzame dve):

```
pH = float(input("Zadaj pH: "))
if pH<0 or pH>14:
   print("Nedovolená hodnota pH!")
else:
   if pH < 7:
      print("kyslý")
   elif pH == 7:
      print("neutrálny")
```

```
pH = float(input("Zadaj pH: "))
if 0 <= pH < 7:
   print("kyslý")
elif pH == 7:
   print("neutrálny")
elif 7 < pH <= 14:
   print("zásaditý")
else:
   print("Nedovolená hodnota pH!")
```

```
else:
    print("zásaditý")
```

Príkaz if možno použiť aj v konštrukcii výraz1 if podmienka else výraz2, ktorý vráti hodnotu výrazu1, ak je podmienka splnená, inak vráti hodnotu výrazu2.

Použitie:

nájdenie maxima z dvoch hodnôt

```
a = -5
b = 4
max = a if a > b else b
print("Väčšie:", max)
```

vypíše

Väčšie: 4

vyhodnotenie prospel/neprospel

```
znamka = 4
hodnotenie = "prospel" if znamka < 5
else "neprospel"
print(hodnotenie)
```

vypíše

prospel

porovnanie dvoch reťazcov

```
s1 = "áno"
s2 = "ano"
prvy = s1 if s1 < s2 else s2
druhy = s1 if s1 > s2 else s2
print(prvy," <= ", druhy)
```

vypíše

ano <= áno

Cykly

Ďalšou riadiacou štruktúrou je cyklus. Používa sa, ak sa má nejaká skupina príkazov opakovane vykonávať. Počet opakovaní môže byť pevne daný alebo ho nepoznáme, a potom je počet opakovaní riadený podmienkou (pokiaľ je splnená, príkazy v cykle sa opakujú, ak nie je splnená, cyklus sa ukončí).

For-cyklus

Príkaz for používame pri pevnom počte opakovaní príkazov v cykle, čo znamená, že poznáme, koľkokrát sa má blok príkazov zopakovať . Varianty for-cyklu je najjednoduchšie predviesť na príkladoch a potom zovšeobecniť.

```
for cislo in 1,2,3,4,5:
    print(cislo, end=" ")
```

1 2 3 4 5

```
for prvok in 1,3,-2,"Fero",'X':
    print(prvok, end=" ")
```

1 3 -2 Fero X

```
for znak in "A","B","E","C","E","D","A":
    print(znak, end="")
```

ABECEDA

```
for znak in "abeceda":
    print(znak, end=" ")
```

a b e c e d a

```
entica = ("Jano", "Fero", "Dušan", "Zuzana")
for prvok in entica:
    print(prvok, end=", ")
```

Jano, Fero, Dušan, Zuzana,

```
zoznam = ["Jano", "Fero", "Dušan", "Zuzana"]
for meno in zoznam:
    print(meno, end=", ")
```

Jano, Fero, Dušan, Zuzana,

```
for i in 0,1,2:
    print(zoznam[i], end=", ")
else: print(zoznam[-1])
```

Jano, Fero, Dušan, Zuzana

Z ukážok vidieť, že za „in“ zapisujeme sekvenciu - rad vecí, nasledujúcich za sebou v istom poradí, postupnosť zložiek tvoriacich celok; odborne hovoríme, že za „in“ zapisujeme iterovateľný objekt (iterácia = opakovanie; iterovateľný objekt - štruktúrovaný objekt, po zložkách ktorého možno prechádzať cyklom; má prvý prvok, druhý prvok atď.). Iterovateľný objekt môže byť zapísaný vymenovaním prvkov (1, 2, 3, 4, 5; 1, 3, -2, "Fero", "X"; "A", "B",..., "abeceda") alebo svojim názvom (entica, zoznam,...). Z ostatnej ukážky vidieť, že môže byť daný aj rozsah indexu iterovateľného objektu a pomocou neho pristupovať k prvkom objektu. Ak pracujeme s väčším počtom prvkov, je nepraktické resp. časovo neúnosné vypisovať všetky hodnoty dovoleného indexu. Preto sa používa generátor čísel, funkcia **range**(). Funkcia range(od, po, krok) má tri parametre, od určuje prvú hodnotu generovanej postupnosti celých čísel, po určuje hodnotu, ktorú už generovaná postupnosť nebude obsahovať (!) a krok je celé číslo udávajúce, o koľko sa majú meniť hodnoty postupnosti. Teda generované hodnoty budú z intervalu <od; po). Povinný je len parameter po, ak chýba parameter od, automaticky sa použije od = 0, ak chýba parameter krok, automaticky sa použije krok = 1.

Ukážka:

```
for cislo in range(1,6):
   print(cislo, end=" ")
```

1 2 3 4 5

```
sucet = 0
for cislo in range(1,101):
   sucet += cislo
print(sucet)
```

5050

V generátore je povinný len parameter po, ak nie je uvedené od, dosadí sa 0, ak nie je daný krok, dosadí sa 1.

```
entica = ("Jano", "Fero", "Dušan", "Peter")
for i in range(len(entica)):
   print(entica[i], end=" ")
```

alebo:

```
for prvok in entica:
   print(prvok, end=" ")
```

Jano Fero Dušan Peter

```
for i in range(len(entica), 2):
   print(entica[i], end=" ")
```

Jano Dušan

```
s = "abeceda"
for i in range(len(s)-1):
  print(s[i], end=", ")
else:
   print(s[-1])
```

a, b, e, c, e, d, a

```
for i in range(90, 64, -1):
   print(chr(i), end=" ")
```

Z Y X W V U T S R Q P O N M L K J I H G F E D C B A

Príkaz for môže mať tvary:

- for *element_iterovateľného_objektu* in *iterovateľný objekt*:
  opakujúci_sa_blok_príkazov
  else:
  blok_príkazov

Napríklad:

```
retazec = "..."
for znak in retazec:
   ... spracuj znak ...                # znak nadobúda hodnoty retazec[0], retazec[1],..., retazec[len(retazec)-1]
```

- for *premenná_cyklu* in *iterovateľný_objekt*:
    opakujúci_sa_blok_príkazov
  else:
    blok_príkazov

Napríklad:

```
sucet = 0
for i in range(1, 10, 2):
   sucet += i                # i nadobudne postupne hodnoty 1,3,5,7,9; cyklus sčíta všetky nepárne celé čísla od 1 po 9
```

- for _ in *iterovateľný_objekt*:
    opakujúci_sa_blok_príkazov
  else:
    blok_príkazov

  Podčiarnik (_) v takýchto cykloch „vystupuje“ ako špeciálna dočasná premenná cyklu.

Napríklad:

```
for _ in range(10):
   print("*", end="")                # vypíše v riadku 10 hviezdičiek
else: print()
```

Vetva else je nepovinná a vykoná sa len raz, na záver for-cyklu (pri použití break alebo return vo funkcii sa preskočí aj vetva else).

Vykonanie: opakujúci_sa_blok_príkazov sa vykonáva, pokiaľ sa neprejdú všetky určené prvky (elementy) iterovateľného objektu. K ukončeniu cyklu dôjde po vyčerpaní všetkých určených prvkov (prípadne príkazmi break, return alebo vznikom neošetrenej výnimky).

Na neštandardné riadenie cyklov (for aj while) sa používajú dva príkazy, a to break a continue.
Príkaz **break** ukončí vykonávanie cyklu a pokračuje sa za príkazom cyklu, v ktorom bol príkaz break použitý.
Napríklad:

```
zoznam = ["Ján","Peter","Pavol","Zuzana","Eva"]
hladat = input("Hľadať meno: ")
for meno in zoznam:
   if meno == hladat:
      print("Meno",hladat,"sa našlo v zozname!")
      break
else: print(hladat,"sa v zozname nevyskytuje!")
```

Zložitejší príklad:

```
for i in range(1,6):
   for j in range(1,6):
      if j%2 == 0: break
      print(i,j, end=", ")
```

vypíše 1 1, 2 1, 3 1, 4 1, 5 1,

Príkaz **continue** spôsobí preskočenie ostatných príkazov v opakujúcom sa bloku príkazov a pokračuje v cykle nasledujúcim prvkom iterácie.

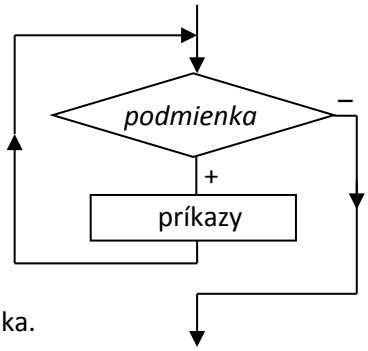
Napríklad

```
for i in range(1,6):
    for j in range(1,6):
        if j%2 == 0: continue
        print(i,j, end=", ")
```

1 1, 1 3, 1 5, 2 1, 2 3, 2 5, 3 1, 3 3, 3 5, 4 1, 4 3, 4 5, 5 1, 5 3, 5 5,

Poznámka: Vykonávanie cyklus for ukončí aj príkaz return - pozri kapitolu Vlastné funkcie.

While-cyklus



musíme použiť, ak nepoznáme počet opakovaní príkazov v cykle.

Má tvar:

```
while výraz_typu_boolen:
        opakujúci_sa_blok_príkazov
else:
        blok_príkazov
```

pričom vetva else je nepovinná a vykoná sa len raz, po ukončení cyklu while.

Na obrázku vpravo je schéma while-cyklu bez vetvy else, ktorá sa používa len zriedka.

Vykonanie: pokiaľ výraz_typu_boolean nadobúda hodnotu True, opakovane sa vykonáva opakujúci_sa_blok_príkazov. Ak výraz_typu_boolean nadobudne hodnotu False, cyklus sa ukončí a ak obsahuje vetvu else, vykoná sa ešte blok_príkazov za else. Ak výraz_typu_boolen nadobudol už pri prvom vyhodnotení hodnotu False, opakujúci_sa_blok_príkazov sa nevykoná ani raz.

Ukážka:

```
cislo = 0
while cislo < 10:
    print(cislo, end=", ")
    cislo += 1
```

vypíše 0, 1, 2, 3, 4, 5, 6, 7, 8, 9,

rovnako ako príkaz for cislo in range(10): print(cislo, end=", ")

Aké zradné je poradie výrazov v podmienkach si ukážeme na nasledujúcej ukážke:

```
zoznam = ["Ján","Dušan","František","Dušan","Zuzana"]
hladat = input("Hľadať meno: ")
i = 0
while i < len(zoznam) and zoznam[i] != hladat:
    i += 1
if i == len(zoznam):
    print("Meno",hladat,"sa v zozname nenachádza!")
else:
    print("Prvý výskyt mena",hladat,"je na indexe",i)
```

N E P O U Ž Í V A Ť !

Program pracuje správne, pokiaľ však zameníme výrazy okolo and[1] a dáme hľadať meno, ktoré sa v zozname nevyskytuje, nastane výnimka IndexError. Pozor na výrazy, ktoré pracujú s indexovanými premennými (zápis premenná[indexový_výraz]), ich hodnota neexistuje, ak hodnota indexového_výrazu bude vyhodnotená mimo dovolený rozsah! **Použitiu indexovaných premenných v podmienkach cyklov je najlepšie sa vyhnúť**.

[1] Problém súvisí s postupom pri vyhodnocovaní zloženej podmienky. Ak vývojové prostredie je nastavené tak, že ak po vyhodnotení prvej časti zloženej podmienky je už zrejmý výsledok výrazu, vo vyhodnocovaní sa ďalej nepokračuje. Vyhodnocovanie zloženej podmienky môže byť nastavené aj tak, že sa vždy vyhodnocuje celý výraz.

„Múdrejšie“ riešenia:

```
zoznam = ["Ján","Dušan","František","Dušan","Zuzana"]
hladat = input("Hľadať meno: ")
i = 0
while i < len(zoznam):
    if zoznam[i] == hladat:
        print("Prvý výskyt mena",hladat,"je na indexe",i)
        break
    i += 1
else:
    print("Meno",hladat,"sa v zozname nenachádza!")
```

alebo

```
zoznam = ["Ján","Dušan","František","Dušan","Zuzana"]
hladat = input("Hľadať meno: ")
for i in range(len(zoznam)):
    if zoznam[i] == hladat:
        print("Prvý výskyt mena",hladat,"je na indexe",i)
        break
else: print("Meno",hladat,"sa v zozname nenachádza!")
```

„Pythonovské“ riešenie bez cyklu s využitím operátora príslušnosti in a funkcie index:

```
zoznam = ["Ján","Dušan","František","Dušan","Zuzana"]
hladat = input("Hľadať meno: ")
if hladat in zoznam:
    print("Prvý výskyt mena",hladat,"je na indexe",zoznam.index(hladat))
else:
    print("Meno",hladat,"sa v zozname nenachádza!")
```

Ako „perličku“ uvádzame riešenie s funkciou enumerate, ako ukážku ďalšieho možného tvaru výrazu vo for-cykle (napriek tomu, že nepracujeme s indexovou premennou zoznam[i], získame index):

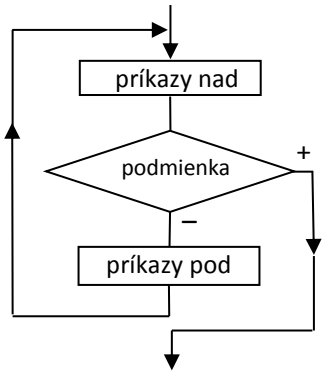
```
zoznam = ["Ján","Dušan","František","Dušan","Zuzana"]
hladat = input("Hľadať meno: ")
for i, meno in enumerate(zoznam):
    if meno == hladat:
        print("Prvý výskyt mena",hladat,"je na indexe",i)
        break
if meno != hladat:
    print("Meno",hladat,"sa v zozname nenachádza!")
```

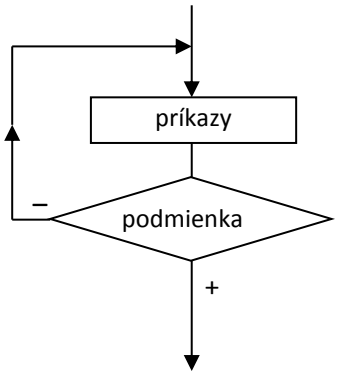
Ako vidíte, riešenie toho istého problému môže mať viac riešení, od všeobecného - použiteľného skoro v každom programovacom jazyku, až po špecializované, využívajúce individuálne vlastnosti jazyka Python.
Čo sa deje vo vašom cykle, si môžete odsimulovať na stránke
http://www.pythontutor.com/visualize.html#mode=edit

Pomocou príkazu while možno modelovať viacej druhov cyklov s podmienkou. V klasickom ponímaní cyklov existujú totiž cyklus s podmienkou na začiatku, cyklus s podmienkou v strede (hovorí sa mu aj úplný cyklus) a cyklus s podmienkou na konci.

Cyklus s podmienkou na začiatku sme si už predstavili (pozri while-cyklus vyššie).

Graficky cyklus s podmienkou v strede a cyklus s podmienkou na konci možno znázorniť:





Keďže v Pythone máme k dispozícii len jeden príkaz cyklu s podmienkou, a to while, musíme zvyšné dva cykly namodelovať pomocou neho.

Cyklus s podmienkou v strede:

```
while True:
   príkazy nad
   if podmienka: break
   príkazy pod
```

Cyklus s podmienkou na konci:

```
while True:
   príkazy
   if podmienka: break
```

V týchto cykloch, na rozdiel od cyklu s podmienkou na začiatku, sa príkazy nad, resp. príkazy, vykonajú vždy aspoň raz. Úlohou hodnoty True v podmienke while-cyklu je vyrobiť nekonečný cyklus ktorý sa ukončuje podmienkou príkazu if!

A čo alternatívy (viete ich graficky znázorniť?):

```
while True:
   príkazy nad
   if podmienka:
      príkazy
      break
   príkazy pod
```

alebo

```
while True:
   príkazy nad
   if podmienka:
      break
   else:
      príkazy pod
```