Podmínečné opakování

Jiří Zacpal

KATEDRA INFORMATIKY
UNIVERZITA PALACKÉHO V OLOMOUCI

KMI/ZPP1 Základy programování v Pythonu 1

Řešení minulého úkolu

```
n=100
prvocislo1=-10
prvocislo2=-10
for c in range(1,n+1):
    pocet_delitelu=0
    for i in range(1,c+1):
        if c % i == 0:
            pocet_delitelu= pocet_delitelu + 1
    je_prvocislo = pocet_delitelu == 2
    if (je_prvocislo):
        prvocislo2=prvocislo1
        prvocislo1=c
        if((prvocislo1-2)==prvocislo2):
            print(prvocislo2,prvocislo1)
```

Řešení minulého úkolu

```
n = 100
for x in range(3, n + 1):
    prvocislo1=True
    prvocislo2=True
    for i in range(2, x):
        if x % i == 0:
            prvocislo1=False
        if (x + 2) % i == 0:
            prvocislo2=False
    if prvocislo1 and prvocislo2:
        print(x, x + 2)
```

Řešení minulého úkolu

```
for i in range(3, 100):
  for j in range(2,i):
    if (i % j) == 0 or ((i + 2) % j) == 0:
      break
  else:
    print(f'{i} {i + 2}')
```

Příkaz přiřazení

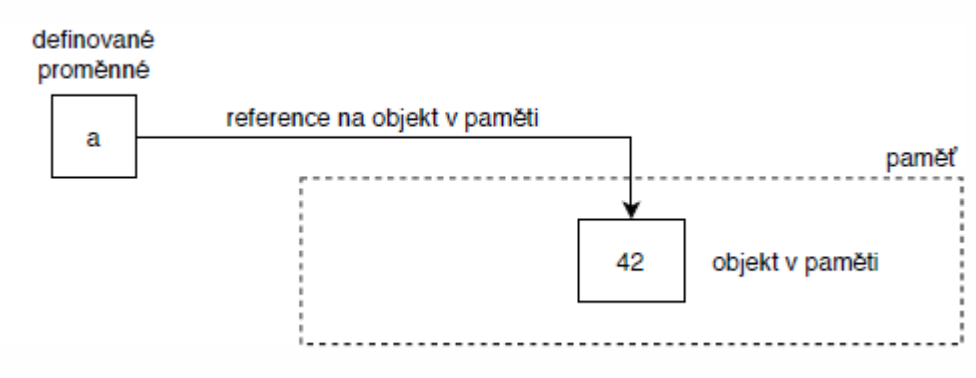
Přiřazení

- Obecný zápis:

L = R

- L – místo, kam lze uložit hodnotu (nejčastěji proměnná).
- R – výraz, jehož hodnota se po vyhodnocení uloží do paměti a odkaz (reference) na ni se uloží do L.
- Příklad:

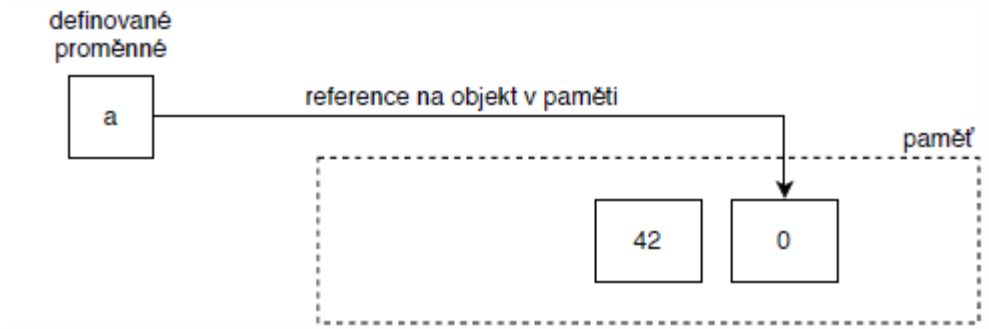
```
a=42
b=a
c=24*a+18
```



Přiřazení a proměnná

- Proměnná, která doposud nebyla použita, se označuje jako nedefinovaná proměnná.
- Pokud je proměnná použita jako levý operand operátoru přiřazení poprvé, dochází k její definici.
- V jazyce Python dochází vždy při definici proměnné i k její deklaraci. Deklarace označuje přiřazení reference k proměnné, tedy přiřazení hodnoty.
- V jazyce Python nelze vytvořit proměnou bez hodnoty.
- První nastavení hodnoty proměnné se také označuje jako inicializace proměnné.
- Pokud je proměnná použita jako levý operand operátoru přiřazení již deklarována, dochází ke změně její reference.
- Příklad:

```
a=42
a=0
```



Vlastnosti přiřazení

- Přiřazení není operátor (jako např. v jazyce C), ale příkaz.
- Tyto příkazy způsobí chybu:

```
>>> a=20+b=10
  File "<stdin>", line 1
SyntaxError: can't assign to operator
>>> (a=10)+(b=2)
  File "<stdin>", line 1
    (a=10)+(b=2)
      ^
SyntaxError: invalid syntax
```

- Lze ale použít příkaz:

```
>>> a=b=10
```

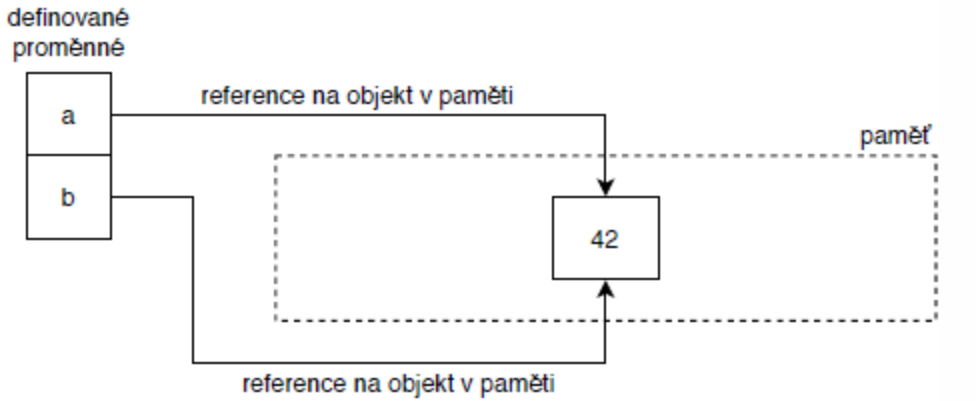
  - Důvodem je, že tento typ zápisu představuje syntaktický cukr, tedy alternativní, pohodlnější zápis jiného výrazu.

Změna reference

- Po vykonání příkazů:

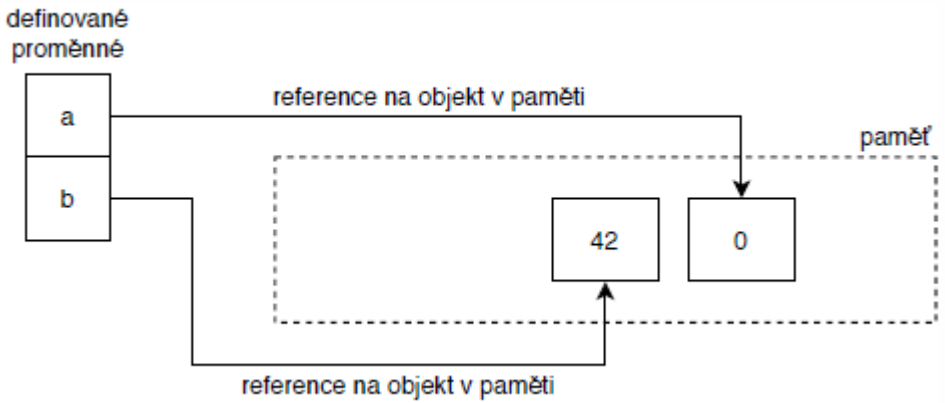
  a=42

  b=a



- Změna nastane po vykonání příkazu:

  a=0



Kombinace s aritmetickými operátory

- Přiřazení lze kombinovat s aritmetickými operátory:

`a+=10` stejný význam jako `a=a+10`

- Kombinace
  - +=
  - -=
  - *=
  - /=
  - %=
  - //=
  - **=

Podmínečné opakování

Příklad

- Chceme vytisknout jednotlivé číslice tříciferného čísla:

```
n = 123
n1 = n

c = n1 % 10
n1 = n1 // 10
print(c)

c = n1 % 10
n1 = n1 // 10
print(c)

c = n1 % 10
n1 = n1 // 10
print(c)
```

Příklad

- Použijeme rozšířený příkaz přiřazení:

```
n = 123
n1 = n

c = n1 % 10
n1 //= 10
print(c)

c = n1 % 10
n1 //= 10
print(c)

c = n1 % 10
n1 //= 10
print(c)
```

Příklad

- Použijeme cyklus for:

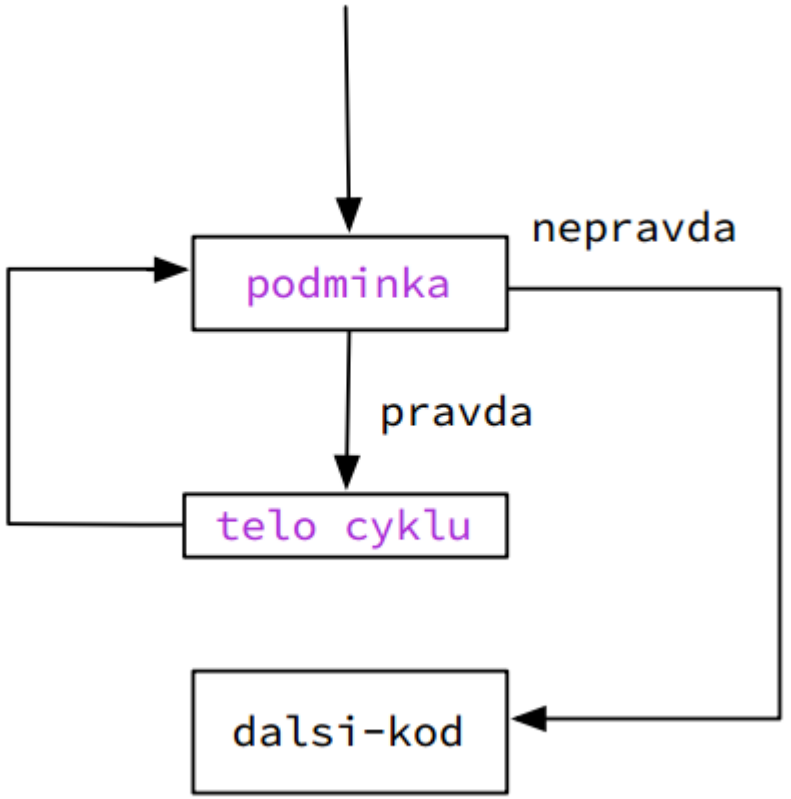
```
n = 123
n1 = n
for i in range(3):
    c = n1 % 10
    n1 //= 10
    print(c)
```

- Co ale když chceme vytisknout jednotlivé číslice obecně n-ciferného čísla a předem nevíme, kolik je n?

Cyklus while

- syntaxe:

```
while (podminka):
    telo cyklu
```



Příklad

- Použijeme cyklus while na předchozí úkol:

```
n = 123
n1 = n

while n1 != 0:
    c = n1 % 10
    n1 //= 10
    print(c)
```

Nekonečný cyklus

- Cyklus, který nikdy neskončí:

```
while True:
    print(0)
```

- Ne vždy je pád do nekonečné smyčky takto průzračný:

```
n = 10
n1 = n
while n1 != 0:
    print(n1)
    n1 -= 2
```

- Co ale když n bude 9?
- Chyba je v podmínce. Jak ji upravíme?

```
while n1 >= 0:
```

Kontrola platnosti vstupu (nezáporné číslo)

- Někdy však zdánlivě nekonečný cyklus požadujeme:

```
a=int(input("Zadej číslo: "))
while (a < 0):
  a=int(input("Zadej číslo znovu: "))
print(f"Zadal jste číslo: {a}")
```

Přepis cyklu for pomocí while

- Vezměme zápis cyklu for:

```
for i in range(v):
    příkazy
```

- Můžeme přepsat pomocí while:

```
n = v
i = 0
while i < n:
    příkazy
    i += 1
```

- Cyklus while nemůžeme obecně přepsat na cyklus for a to z toho důvodu, že cyklus for narozdíl od cyklu while vždy skončí.

Příklad

- Vraťme se k určování prvočísla:

```
n = 7
je_prvocislo = True
for i in range(2,n):
    if n % i == 0:
        je_prvocislo= False
print(je_prvocislo)
```

- Cyklus celkem zbytečně probíhá, i když už je je_prvočíslo False. Můžeme použít break, ale lepší je použít cyklus while:

```
n = 7
je_prvocislo = True
i=2

while i<n and je_prvocislo==True:
    if n % i == 0:
        je_prvocislo= False
    i+=1

print(je_prvocislo)
```

Úkol

- Rozhodněte, zda je libovolné přirozené číslo palindrom.

- Řešení:

```
cislo=1221
i=1
p=0
l=0
pocet=0

k=cislo
while(k>=1):
    k//=10
    pocet+=1
pocet//=2
```

Úkol

```
while(l==p and i<=pocet):
    k=cislo
    for j in range(1,i):
        k//=10
    p=k%10

    k=cislo
    while (k>10**i):
        k//=10
    l=k%10

    i+=1

if(l==p):
    print("Je palindrom")
else:
    print("Není palindrom")
```

Přerušení iterace

Příklad

- Někdy by bylo výhodné přerušit iteraci způsobenou příkazem cyklu for:

```
n = 7

je_prvocislo = True
for i in range(2,n):
    if n % i == 0:
        je_prvocislo= False

print(je_prvocislo)
```

Příkazy přerušení cyklu

- příkaz `continue`
  - skok na konec nejvnitřnějšího cyklu, výpočet pokračuje další iterací (včetně testu případné podmínky)
- příkaz `break`
  - okamžité opuštění nejvnitřnějšího cyklu
- Vztahují se vždy k „nejbližšímu“ cyklu

Příklad

- Někdy by bylo výhodné přerušit iteraci způsobenou příkazem cyklu for:

```
n = 7

je_prvocislo = True
for i in range(2,n):
    if n % i == 0:
        je_prvocislo= False
        break

print(je_prvocislo)
```

- Příkaz break se musí nacházet v těle cyklu:

```
>>> break
  File "<stdin>", line 1
SyntaxError: 'break' outside loop
>>>
```

Příklad

- Vezměme si nyní následující program, který tiskne dvojice nezáporných čísel menších než zadané číslo:

```
n = 10
for i in range(n):
    for j in range(n):
        print(i, j)
```

- Co když budeme chtít program upravit tak, aby první číslo bylo menší nebo rovno než druhé číslo:

```
n = 10
for i in range(n):
    for j in range(n):
        if i<=j:
            print(i, j)
```

- Není to efektivní.

Příklad

- Program upravíme takto:

```
n = 10
for j in range(n):
    for i in range(n):
        if j < i:
            break
        print(i, j)
```

- Dá se program napsat bez break?

```
n = 10
for j in range(n):
    for i in range(j + 1):
        print(i, j)
```

Příklad

- Nalezení x-tého čísla dělitelného číslem n, které je větší nebo rovno číslu od. Navíc vypisujeme všechna testovaná čísla, která nejsou dělitelná číslem n.

```
n=int(input("Zadej číslo n: "))
x=int(input("Zadej číslo x: "))
od=int(input("Zadej číslo od: "))

nalezeno=0
i=od-1
while (True):
    i+=1
    if (i%n==0):
        nalezeno+=1
        if (nalezeno==x):
            break
        else:
            continue
    print(f"{i},",end=" ")

print(f"\n{x}-té číslo dělitelné {n} od čísla {od} je číslo {i}\n")
```

Úkol

- Rozhodněte, zda jsou dvě čísla nesoudělná. Ukončete program, jakmile zjistíte netriviálního dělitele.

- Řešení:

```
x=2
y=4
if (x<y):
    n=x
else:
    n=y

i=2

while(i<=n):
    if(x%i==0 and y%i==0):
        break
    else:
        i+=1

if(i>n):
    print("Čísla jsou nesoudělná.")
else:
    print("Čísla jsou soudělná.")
```

Bodovaný úkol

- Vytisknete rozklad přirozeného čísla na prvočísla.

- **Příklad výstupu:**

```
Prvočíselný rozklad čísla 36 je:
2 2 3 3
```