Wstęp do programowania

Pętle

Paweł Daniluk

Wydział Fizyki

Jesień 2014

Przypomnienie

Po co?

Trzeba wykonać tę samą (lub podobną) czynność wielokrotnie.

Szczegóły

- Co wykonać?
- Ile razy?
- Co się zmienia w kolejnych iteracjach?

Co wykonać?

Przykłady

- Wypełnianie/aktualizacja elementów

```
for i in range(10):
    tab[i] = i * i
```

Co wykonać?

Przykłady

- Wypełnianie/aktualizacja elementów

```
for i in range(10):
    tab[i] = i * i
```

- Zbieranie statystyk

```
suma = 0
for el in lista:
    suma = suma + el
```

Co wykonać?

Przykłady

- Wypełnianie/aktualizacja elementów

```
for i in range(10):
    tab[i] = i * i
```

- Zbieranie statystyk

```
suma = 0
for el in lista:
    suma = suma + el
```

- Wyszukiwanie

```
for el in lista:
    if el == wzorzec:
        break
```

Ile razy?

- W nieskończoność

```
while True:
    l = input("Podaj liczbe: ")
    if l%2>0:
        print "Nieparzysta"
    else:
        print "Parzysta"
```

Ile razy?

- W nieskończoność

```
while True:
    l = input("Podaj liczbe: ")
    if l%2>0:
        print "Nieparzysta"
    else:
        print "Parzysta"
```

- Ustaloną liczbę razy

```
a = 1
for i in range(10):      # a = 2 ** 10
    a = a * 2
```

Przerywanie pętli

- Dopóki jest spełniony warunek

```
log = 0
while a > 1:     # logarytm dla ubogich
    a = a / 2
    log = log + 1
```

Przerywanie pętli

- Dopóki jest spełniony warunek

```
log = 0
while a > 1:     # logarytm dla ubogich
    a = a / 2
    log = log + 1
```

- Dla wszystkich elementów listy

```
for el in lista:
    if el == wzorzec:
        break
```

Przerywanie pętli

- Dodatkowy warunek końca

```
for el in lista:
    if el == wzorzec:
        break # znaleziono
```

Przerywanie pętli

- Dodatkowy warunek końca

```
for el in lista:
    if el == wzorzec:
        break # znaleziono
```

- Obsługa sytuacji wyjątkowej

```
while True:
    l = input("Podaj liczbe (zero konczy prace): ")

    if l == 0:
        break

    if l%2 > 0:
        print "Nieparzysta"
    else:
        print "Parzysta"
```

Przerywanie iteracji - instrukcja kontynuacji

```
for i in range(10):
    if i%2 == 0:
        if sin(i) > 0:
            print i, ' jest parzysta liczba o dodatnim sinusie.'
```

```
for i in range(10):
    if i%2 != 0:
        continue
    if sin(i) <= 0:
        continue

    print i, ' jest parzysta liczba o dodatnim sinusie.'
```

W ten sposób można pominąć iteracje dla pewnych wartości licznika. To może być łatwiejsze rozwiązanie, niż iterowanie od razu po mniejszym zbiorze.

Co zmienia się w kolejnych iteracjach?

- Nic

```
while True:
    l = input("Podaj liczbe: ")
    if l%2>0:
        print "Nieparzysta"
    else:
        print "Parzysta"
```

Co zmienia się w kolejnych iteracjach?

- Nic

```
while True:
    l = input("Podaj liczbe: ")
    if l%2>0:
        print "Nieparzysta"
    else:
        print "Parzysta"
```

- Wartości zmiennych (stan programu)

```
log = 0
while a > 1:     # logarytm dla ubogich
    a = a / 2
    log = log + 1
```

Co zmienia się w kolejnych iteracjach?

- Nic

```
while True:
    l = input("Podaj liczbe: ")
    if l%2>0:
        print "Nieparzysta"
    else:
        print "Parzysta"
```

- Wartości zmiennych (stan programu)

```
log = 0
while a > 1:     # logarytm dla ubogich
    a = a / 2
    log = log + 1
```

- Licznik pętli

Licznik pętli

- Zazwyczaj liczba naturalna.
- Zazwyczaj przebiega od zera do pewnej wartości.
- Zazwyczaj rośnie, ale może maleć.

W pętli `while`

```
i = od
while i < do:
    ....
    i = i + krok
```

W pętli `for`

```
for i in range(od, do, krok):
    ...
```

Zachowanie powyższych programów może się różnić, jeżeli w ... zmieniana będzie wartość `i`.

Akumulator

Zmienna, która w kolejnych iteracjach jest modyfikowana, aby po wykonaniu pętli osiągnąć wymaganą wartość.

$f(1) + f(2) + \ldots + f(10) = \sum_{i=1}^{10} f(i)$

```
suma = 0
for i in range(1, 11):
    suma = suma + f(i)
```

Nie wolno zapominać o ustawieniu początkowej wartości akumulatora. Powinna to być wartość niezmiennicza dla wykonywanej operacji (np. 0 dla dodawania i 1 dla mnożenia).

Pętle zagnieżdżone

Po co?

- Iteracja po przestrzeni wielowymiarowej

```
for i in range(10):
    for j in range(10):
        tab[i][j] = i * j
```

Działanie wewnętrznej pętli może, ale nie musi zależeć od pętli zewnętrznej.

Pętle zagnieżdżone

Po co?

- Iteracja po przestrzeni wielowymiarowej

```
for i in range(10):
    for j in range(10):
        tab[i][j] = i * j
```

- Składanie operatorów (np. $1! + 2! + \ldots 10! = \sum_{i=0}^{10} \prod_{j=1}^{i} j$)

```
suma = 0
for i in range(10):
    prod = 1
    for j in range(1, i+1):
        prod = prod * j
    suma = suma + prod
```

Działanie wewnętrznej pętli może, ale nie musi zależeć od pętli zewnętrznej.

Liczba iteracji w pętlach zagnieżdżonych

Jeżeli liczba iteracji w wewnętrznej pętli nie zależy od iteracji zewnętrznej, to łącznie zostanie wykonanych $n \times m$ iteracji PW (gdzie n i m są odpowiednio liczbami iteracji PZ i PW).

```
licznik = 0
for i in range(10):
    for j in range(10):
        licznik = licznik + 1

# tutaj licznik == 100
```

W ogólnym przypadku jest trudniej.

```
licznik = 0
for i in range(10):          # 10 iteracji
    for j in range(i):       # i iteracji
        licznik = licznik + 1
```

Trudna uwaga o pętli `while`

Projektując pętlę można określić niezmiennik (N) i warunek końca pętli (P). Jeżeli przed rozpoczęciem iteracji jest N jest spełniony i kolejne iteracje go nie naruszają, to po zakończeniu pętli mamy $N \wedge \neg P$.

Trudna uwaga o pętli `while`

Projektując pętlę można określić niezmiennik (N) i warunek końca pętli (P). Jeżeli przed rozpoczęciem iteracji jest N jest spełniony i kolejne iteracje go nie naruszają, to po zakończeniu pętli mamy $N \wedge \neg P$.

```
suma = 0
i = 0
while i != n :    # powinno byc i < n
    suma  = suma + f(i)
    i = i + 1
```

Trudna uwaga o pętli `while`

Projektując pętlę można określić niezmiennik (N) i warunek końca pętli (P). Jeżeli przed rozpoczęciem iteracji jest N jest spełniony i kolejne iteracje go nie naruszają, to po zakończeniu pętli mamy $N \wedge \neg P$.

```
suma = 0
i = 0
while i != n :    # powinno byc i < n
    suma  = suma + f(i)
    i = i + 1
```

$$N = \left\{ suma = \sum_{k=0}^{i-1} f(k) \right\}$$

$$P = \{i \neq n\}$$

Trudna uwaga o pętli `while`

Projektując pętlę można określić niezmiennik (N) i warunek końca pętli (P). Jeżeli przed rozpoczęciem iteracji jest N jest spełniony i kolejne iteracje go nie naruszają, to po zakończeniu pętli mamy $N \wedge \neg P$.

```
suma = 0
i = 0
while i != n :    # powinno byc i < n
    suma  = suma + f(i)
    i = i + 1
```

$$N = \left\{ suma = \sum_{k=0}^{i-1} f(k) \right\}$$

$$P = \{ i \neq n \}$$

$$N \wedge \neg P \Longrightarrow suma = \sum_{k=0}^{i-1} f(k) \wedge i = n \Longrightarrow suma = \sum_{k=0}^{n-1} f(k)$$

Zadania

1. Generowanie ciągu N losowych liczb całkowitych.
2. Generowanie monotonicznie rosnącego ciągu liczb N całkowitych.
3. Wyszukiwanie binarne.
4. Scalanie dwóch uporządkowanych list.
5. Sortowanie bąbelkowe.
6. Sortowanie przez wstawianie.
7. Flaga polska.
8. Segment o maksymalnej sumie.

Zadania

1. Generowanie ciągu N losowych liczb całkowitych.
2. Generowanie monotonicznie rosnącego ciągu liczb N całkowitych.
3. Wyszukiwanie binarne.
4. Scalanie dwóch uporządkowanych list.
5. Sortowanie bąbelkowe.
6. Sortowanie przez wstawianie.
7. Flaga polska.
8. Segment o maksymalnej sumie.

Generator liczb pseudolosowych

```
>>> import random
>>> random.randint(0,1)
0
>>> random.randint(0,1)
1
>>> random.randint(0,1)
0
```

Strona wykładu

http://bioexploratorium.pl/wiki/Wstęp_do_programowania_
_2014z