

Zasady dobrego programowania

Dawid Kurzępa

WMiFS IIAD I L4

AiSD Projekt
prowadzący:

dr inż. Mariusz
Borkowski

Prezentacja nie ma
na celu nikogo
urazić/zniechęcić.

**Od czegoś trzeba
zacząć!**

Komentarze

Tak nie robimy:

Ktoś „obeznany” w temacie prawdopodobnie poradzi sobie z analizą kodu jednak jest to tylko strata czasu.

```
import random
N = 100
D = []
for _ in range(N):
    liczba = random.randint(0, 49)
    D.append(liczba)
def quicksort(tablica):
    if len(tablica) <= 1:
        return tablica
    pivot = tablica[len(tablica) // 2]
    mniejsze = [x for x in tablica if x < pivot]
    rowne = [x for x in tablica if x == pivot]
    wieksze = [x for x in tablica if x > pivot]
    return quicksort(mniejsze) + rowne + quicksort(wieksze)
print("Tablica przed sortowaniem:")
print(D)
D = quicksort(D)
print("\nTablica po sortowaniu:")
print(D)
```

Komentarze

Jak to powinno wyglądać:

Nawet, jeżeli ktoś nie zna języka programowania python i tak może przeanalizować ten kod. Każda linia kodu jest opisana – za co odpowiada.

Pamiętajmy że mamy do czynienia z tym samym kodem w obu przypadkach – losowanie do tablicy, quicksort z pivotem i wyświetlenie tablicy.

Niezależnie od języka programowania należy komentować programy.

```
import random # Importujemy moduł random, aby móc losować liczby
# Ustawiamy rozmiar tablicy
N = 100
# Tworzymy pustą tablicę D o rozmiarze N
D = []
# Losujemy 100 liczb pseudolosowych z przedziału 0-49 i dodajemy je do tablicy D
for _ in range(N):
    liczba = random.randint(0, 49) # Losujemy liczbę z przedziału 0-49
    D.append(liczba) # Dodajemy wylosowaną liczbę do tablicy
# Funkcja quicksort
def quicksort(tablica):
    # Jeżeli tablica ma długość 1 lub 0, jest już posortowana
    if len(tablica) <= 1:
        return tablica
    # Wybieramy pivot, tutaj używamy środka tablicy
    pivot = tablica[len(tablica) // 2]
    # Dzielimy elementy tablicy na trzy listy:
    mniejsze = [x for x in tablica if x < pivot] # - mniejsze od pivota,
    rowne = [x for x in tablica if x == pivot] # - większe od pivota.
    wieksze = [x for x in tablica if x > pivot] # - większe od pivota.
    # Rekurencyjnie sortujemy mniejsze i większe elementy, a następnie łączymy
    return quicksort(mniejsze) + rowne + quicksort(wieksze)
# Wyświetlamy tablicę przed sortowaniem
print("Tablica przed sortowaniem:")
print(D)
# Sortujemy tablicę D za pomocą quicksort
D = quicksort(D)
# Wyświetlamy tablicę po sortowaniu
print("\nTablica po sortowaniu:")
print(D)
```

Nazwy zmiennych

Tak nie robimy:

Nazwy zmiennych kompletnie nie odzwierciedlają ich zastosowania i trzeba ręcznie wydedukować do czego służą, brak komentarzy potęguje problem.

```
import random
r = 100
t = []
for _ in range(r):
    l = random.randint(0, 49)
    t.append(l)
def quicksort(t):
    if len(t) <= 1:
        return t
    p = t[len(t) // 2]
    m = [x for x in t if x < p]
    r1 = [x for x in t if x == p]
    w = [x for x in t if x > p]
    return quicksort(m) + r1 + quicksort(w)
print("t przed sortowaniem:")
print(t)
t = quicksort(t)
print("\nt po sortowaniu:")
print(t)
```

Nazwy zmiennych

Jak to powinno wyglądać:

W tym przypadku mimo braku komentarzy widzimy za co odpowiada każda zmienna. Tutaj trzeba zaznaczyć, że dobrą praktyką jest nazywanie zmiennych w języku zrozumiałym dla wszystkich pracujących nad projektem (najczęściej angielski).

```
import random
rozmiar = 100
tablica = []
for _ in range(rozmiar):
    liczba = random.randint(0, 49)
    tablica.append(liczba)
def quicksort(tablica):
    if len(tablica) <= 1:
        return tablica
    pivot = tablica[len(tablica) // 2]
    mniejsze = [x for x in tablica if x < pivot]
    rowne = [x for x in tablica if x == pivot]
    wieksze = [x for x in tablica if x > pivot]
    return quicksort(mniejsze) + rowne + quicksort(wieksze)
print("Tablica przed sortowaniem:")
print(tablica)
tablica = quicksort(tablica)
print("\nTablica po sortowaniu:")
print(tablica)
```

Podział na funkcje

Tak nie robimy:

Mimo bardzo prostego programu wielokrotnie powtarzamy ten sam fragment kodu tj. potęgowanie i wypisanie.

Stworzenie takiego programu w większej skali będzie BARDZO nieefektywne i niewydajne

```
print("Liczba | Kwadrat | Sześciyan")
print("-----")
print(1, " | ", 1**2, " | ", 1**3)
print(2, " | ", 2**2, " | ", 2**3)
print(3, " | ", 3**2, " | ", 3**3)
print(4, " | ", 4**2, " | ", 4**3)
print(5, " | ", 5**2, " | ", 5**3)
print(6, " | ", 6**2, " | ", 6**3)
print(7, " | ", 7**2, " | ", 7**3)
print(8, " | ", 8**2, " | ", 8**3)
print(9, " | ", 9**2, " | ", 9**3)
print(10, " | ", 10**2, " | ", 10**3)
```

Podział na funkcje

Jak to powinno wyglądać:

Ten sam program, który możemy rozszerzyć zmieniając kilka wartości w programie zamiast kopiować poprzednie fragmenty w nieskończoność

```
def potegi(liczba):
    kwadrat = liczba ** 2
    szescian = liczba ** 3
    return kwadrat, szescian

def wypisz_tabelke():
    print("Liczba | Kwadrat | Sześciąn")
    print("-----")
    for liczba in range(1, 11):
        kwadrat, szescian = potegi(liczba)
        print(f"{liczba:<6} | {kwadrat:<7} | {szescian}")

wypisz_tabelke()
```

Przepelenie bufora

Program obrazujacy problem:

Celowo wyszliemy poza zakres
Aby pokazać problem
(w tym przypadku zapisujemy
elementy do nieistniejących – za
duzych indeksów tablicy)

```
Początkowy bufor: [0, 0, 0, 0, 0]
Zapis do bufora[0] = 0
Zapis do bufora[1] = 10
Zapis do bufora[2] = 20
Zapis do bufora[3] = 30
Zapis do bufora[4] = 40
Błąd przepelenienia bufora: list assignment index out of range
Ostateczny bufor: [0, 10, 20, 30, 40]
```

```
def buffer_overflow_simulation():
    # Ustawiamy bufor o rozmiarze 5 elementów
    buffer = [0] * 5

    # Wyświetlamy początkowy stan bufora
    print("Początkowy bufor:", buffer)

    # Próbujemy zapisać wartość poza zakresem bufora (indeks poza 4)
    try:
        for i in range(7): # Celowo wychodzimy poza zakres bufora
            buffer[i] = i * 10
            print(f"Zapis do bufora[{i}] = {i * 10}")
    except IndexError as e:
        print("Błąd przepelenienia bufora:", e)

    # Wyświetlamy ostateczny stan bufora
    print("Ostateczny bufor:", buffer)

# Uruchomienie symulacji
buffer_overflow_simulation()
```

Niepoprawne argumenty

Program obrazujący problem:

Funkcja dodaj pobiera argumenty a oraz b i je sumuje, w tym przypadku programista wywołał funkcję podając tylko jeden argument.

```
def dodaj(a, b):  
    return a + b  
  
# Wywołanie funkcji z błędna liczbą argumentów  
wynik = dodaj(5) # Brakuje drugiego argumentu  
print(wynik)
```

Wynikiem programu będzie następujący błąd:

```
TypeError: dodaj() missing 1 required positional argument: 'b'
```

Wartości większe niż zmienna

Program obrazujący problem:

Celowo ograniczamy zakres zmiennej liczącej wyrazy ciągu Fibonacciego aby szybciej zauważyc problem.

```
def fibonacci_overflow(max_value):
    a, b = 0, 1
    index = 1

    while True:
        # Symulacja ograniczenia zakresu zmiennej
        if b > max_value:
            print(f"Błąd: przekroczeno zakres zmiennej. {b} w indeksie {index}")
            break
        print(f"F({index}) = {b}")
        a, b = b, a + b
        index += 1

# Ustalmy maksymalną wartość zmiennej, aby szybko zauważyc błąd
fibonacci_overflow(100000)
```

```
F(23) = 28657
F(24) = 46368
F(25) = 75025
Błąd: przekroczeno zakres zmiennej. 121393 w indeksie 26
```

Testowanie kodu

Program obrazujący problem:

Zawsze trzeba zakładać najgorsze scenariusze przy kontakcie z klientem przy pisaniu kodu.

Programista testujący kod:

```
Podaj liczbę: 2
Wynik dzielenia 10 przez 2.0 to: 5.0
```

Klient:

```
Podaj liczbę: 0
Traceback (most recent call last):
  File "C:\Users\Kamil\Desktop\Python\test.py", line 11, in <module>
    divide_by_user_input()
  File "C:\Users\Kamil\Desktop\Python\test.py", line 6, in divide_by_user_input
    result = 10 / number
               ~~~~~
ZeroDivisionError: float division by zero
```

```
def divide_by_user_input():
    # Pobranie liczby od użytkownika
    number = float(input("Podaj liczbę: "))

    # Próba podzielenia 10 przez podaną liczbę
    result = 10 / number

    print(f"Wynik dzielenia 10 przez {number} to: {result}")

# Wywołanie funkcji
divide_by_user_input()
```

Dziękuję za uwagę

Pamiętajcie o
komentarzach!!!!!!
(i reszcie też)

Prezentacja jest
dostępna do pobrania
dla wszystkich

