Marzena Nowakowska, Wydział Zarządzania i Modelowania Komputerowego, PŚk

Temat 4. Funkcje projektanta. Obsługa błędów

Przypomnienie definicji z kombinatoryki

Permutacją zbioru *n*-elementowego jest dowolny *n*-wyrazowy ciąg utworzony ze wszystkich elementów tego zbioru. Kolejność elementów permutacji jest istotna. Liczba takich permutacji jest równa:

$P_0 = 0! = 1, \quad P_n = n! = 1 \cdot 2 \cdot \ldots \cdot (n-1) \cdot n$

Kombinacją *k*-elementowa zbioru *n*-elementowego, gdzie *k, n* są liczbami naturalnymi, *k*<=*n*, jest każdy *k*-elementowy podzbiór tego zbioru. Kolejność elementów kombinacji nie jest istotna. Liczba takich kombinacji jest dana wzorem:

$$C_n^k = \frac{n!}{k!(n-k)!}$$ Ww. zależność nosi nazwę symbolu Newtona $$\binom{n}{k} = \frac{n!}{k!(n-k)!}$$

Wariacją *k*-elementową **bez powtórzeń** utworzoną ze zbioru n-elementowego (*k*≤*n*) jest każdy *k*-wyrazowy ciąg różnych elementów z tego zbioru. Kolejność elementów wariacji bez powtórzeń jest istotna. Liczba takich wariacji jest równa:

$$V_n^k = \frac{n!}{(n-k)!}$$

Wariacją *k*-elementową **z powtórzeniami** utworzoną ze zbioru *n*-elementowego jest każdy *k*-wyrazowy ciąg elementów z tego zbioru. Kolejność elementów wariacji z powtórzeniami jest istotna. Liczba takich wariacji jest równa:

$$W_n^k = n^k$$

Zadania do rozwiązania

Zad. 1.

Zdefiniować program *Zad04_01* zawierający funkcję *silnia* obliczającą wartość silni liczby *n*. Wykorzystać ją do zdefiniowania funkcji liczących: kombinację, wariację bez powtórzeń i wariację z powtórzeniami. Przetestować działanie funkcji (można wprowadzić instrukcje wołania funkcji w programie głównym lub konsoli). Zdefiniować drugą wersję rozwiązania, gdzie kombinacja i obie wariacje są wyznaczane w jednej funkcji.

Zad. 2.

W programie zapisano współrzędne wierzchołków wielokąta w dwu listach: X = [X_0 ,X_2,…., X_{n-1}], Y = [Y_0,Y_2,…., Y_{n-1}]. Napisać funkcję *Wlk_p_o*, która zwróci wartość pola wielokąta oraz obwód wielokąta. Do wyznaczenia pola wielokąta wykorzystać wzór Gaussa:

$$P = \frac{1}{2}\left|\sum_{i=0}^{n-1} X_i \cdot (Y_{i+1} - Y_i)\right| = \frac{1}{2}\left|\sum_{i=0}^{n-1} Y_i \cdot (X_{i-1} - X_{i+1})\right|$$

gdzie:

P – pole powierzchni, n – liczba wierzchołków wielokąta, i – nr współrzędnej wierzchołka. Zakłada się, że $X_{-1} = X_{n-1}$, $X_n = X_0$, $Y_{-1}=Y_{n-1}$, $Y_n = Y_0$.

Przykładowe wywołanie: *Pole, Obwod = Wlk_p_o (X, Y)*

Zad. 3.

Zdefiniować w pliku *Zad04_03* program wyznaczający statystyki wyrazów w teksie; przepisać podany niżej program. Wprowadzić komentarze we wskazanych miejscach.

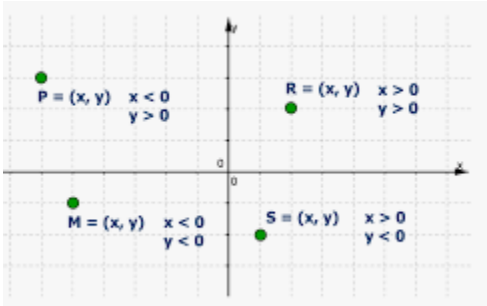
```
def podzialNapisu(tekst):
# Wprowadzić komentarz objaśniający działanie funkcji,
# w tym rolę parametru.
```

```
    wyrazy=tekst.split()
    return wyrazy

def filtrWyrazów(wyrazy, dlg):
# Wprowadzić komentarz objaśniający działanie funkcji,
# w tym role parametrów.
    odfiltrowane=[el for el in wyrazy if len(el)>dlg]
    return odfiltrowane

def czestWyst(wyrazy):
# Wprowadzić komentarz objaśniający działanie funkcji,
# w tym rolę parametru.
    wyrazyUnikatowe = set(wyrazy)
    sttsWrzw={}
    for el in wyrazyUnikatowe:
        klucz=el
        wartosc=wyrazy.count(el)
        sttsWrzw.update({klucz:wartosc})
    return sttsWrzw

tekst="""to jest tekst zawierający różne zdania. Zdania
mogą być lub nie być poprawne gramatycznie i ortograficznie. w szczególności
mogą zaczynać się małą literą.    Ponadto mogą mieć za dużo białych
znaków wiodących lub końcowych. W tekście mogą występować różne znaki interpunkcyjne,
jak np.: kropka, przecinek, wykrzyknik, średnik, dwukropek, pytajnik, apostrof,
nawiasy.
Aby wstępnie przygotować tekst do analiz należy:
- wyodrębnić wyrazy,
- usunąc znaki interpunkcyjne.
"""
znakiSpec=".,!;:?'()[]{}"          # 1 - przypisanie
noweWyrazy=podzialNapisu(tekst)    # 2 - przypisanie
for znak in znakiSpec:             # 3 - petla
    noweWyrazy=[(el.replace(znak,"")).upper() for el in noweWyrazy]
odfiltrowaneWyrazy=filtrWyrazów(noweWyrazy,1)
lczbyWrz = czestWyst(odfiltrowaneWyrazy)
```

1. Objaśnić instrukcje występujące w funkcji *czestWyst*.
2. Objaśnić fragment programu oznaczony numerami: 1, 2, 3.
3. Co jest wynikiem programu?
4. Uzupełnić program o instrukcję wyświetlającą wynik w oknie konsoli.

Zad. 4.

W programie *Zad04_04* definiować funkcję *nrCwiartki*, która zwraca informację o tym, w której ćwiartce leży punkt o współrzędnych *x* i *y*. Program główny bada zbiór, w którym każdy element jest dwuelementową krotką współrzędnych punktu na płaszczyźnie. Wykorzystując funkcję *nrCwiartki* tworzy słownik, w którym klucz jest numerem ćwiartki, a wartością elementu słownika jest liczba punktów leżących w danej ćwiartce. Punktów leżących na osiach układu współrzędnych nie zalicza się do żadnej ćwiartki.



Punktów leżących na osiach układu współrzędnych nie zalicza się do żadnej ćwiartki.

Zad.5.
Wykorzystując informację podaną poniżej, wykonać śledzenie programu *Zad04_04*. Sprawdzić, jakie wartości przyjmują elementy słownika przy każdorazowym przebiegu pętli tworzącej słownik liczników dla zbioru ćwiartek układu współrzędnych.

Sposób postępowania przy śledzeniu programu w środowisku Spyder

1. Włączyć (aktywować) debugger: Ctrl+F5, menu główne: *Debug* /*Debug* lub pasek narzędzi przycisk  . Panel edytora podświetli linię, od której rozpocznie się wykonywanie programu w trybie śledzenia. Eksplorator zmiennych wyświetli zmienne w bieżącym kontekście programu. W oknie konsoli zmieni się treść zachęty z In[*nr*]: na ipdb>
2. Po wejściu w tryb śledzenia można wykonywać program wiersz po wierszu za pomocą menu *Debug*/ *Step*, przycisku  lub Ctrl+F10.
3. Jeżeli razem ze śledzeniem programu ma być śledzone działanie funkcji należy wykorzystać menu *Debug/Step into*, przycisk  lub Ctrl+F11. Można wtedy klikać tylko ten przycisk dla całego programu.
4. Aby opuścić śledzenie funkcji należy wykorzystać menu *Debug/Step return*, przycisk  lub Ctrl+F12 (przydatne w przypadku funkcji wbudowanych).
5. Po zakończeniu procesu śledzenia należy je wyłączyć poprzez menu *Debug/Stop*, przycisk  lub Ctrl+Shift+F12. W ten sposób można przerwać debugowanie programu w dowolnym miejscu.

Zad. 6.
Napisać w pliku *Zad04_06* program obsługi błędów (program z wykładu):

```
try:
   print("\nObliczenie wielkosci zmiany ceny towaru")
   print("=====================================")
   cena_przed=float(input("Podaj cenę towaru przed zmianą: "))
   cena_po=float(input("Podaj cenę towaru po zmianie: "))
   zmiana= -(cena_przed-cena_po)/cena_przed*100
except ValueError:
   print('\nWprowadzony ciąg znaków niezgodny z formatem liczby. Brak wyniku')
   zmiana=None
except ZeroDivisionError:
   print('\nWprowadzona cena przed zmiana nie może być zerem. Brak wyniku')
   zmiana=None
   print('Wprowadzone zero jako dzielnik zastąpiono liczbą 10')
   n=10
except:
   print("\nInny błąd w przetwarzaniu danych. Brak wyniku")
         #np. Ctrl+c (przerwanie operacji), Ctlr+d (znak końca pliku)
   zmiana=None
else:
   print('\nBrak błędów przetwarzaniu danych. Wynik poprawny')
finally:
   print('\nProcent zmiany ceny: ' , end='')
   if zmiana ==None:
      print(zmiana)
   else:
      print("%+6.2f%s" %(zmiana, "%"))
```

1. Uruchomić program kilkakrotnie, wprowadzając za każdym razem takie dane, aby testowane były wszystkie ścieżki; błędy oraz ich brak.
2. Uzupełnić we właściwym miejscu program tak, aby obsługiwany był błąd wprowadzenia danej nieprawidłowego typu (sprawdzić w oknie konsoli, jaka stała identyfikuje taki błąd). Przetestować działanie programu po jego modyfikacji.