Uniwersytet Ekonomiczny w Katowicach
Katedra Inżynierii Wiedzy
Kolegium Informatyki i Komunikacji

Jan Kozak

Zadania z programowania w języku Python

Katowice, 2020 r.

Spis treści

Zadania wstępne – instrukcje warunkowe i pętle

1. Napisać program, który oblicza wartość współczynnika BMI (ang. body mass index) wg. wzoru: $\frac{waga}{\mathrm{w}zrost^2}$. Jeżeli wynik jest w przedziale (18,5 - 24,9) to wypisuje "waga prawidłowa", jeżeli poniżej to "niedowaga", jeżeli powyżej "nadwaga".

2. W sklepie ze sprzętem AGD oferowana jest sprzedaż ratalna. Napisz program umożliwiający wyliczenie wysokości miesięcznej raty za zakupiony sprzęt. Danymi wejściowymi dla programu są:

   - cena towaru (od 100 zł do 10 tyś. zł),
   - liczba rat (od 6 do 48).

   Kredyt jest oprocentowany w zależności od liczby rat:

   - od 6–12 wynosi 2.5% ,
   - od 13–24 wynosi 5%,
   - od 25–48 wynosi 10%.

   Obliczona miesięczna rata powinna zawierać również odsetki. Program powinien sprawdzać, czy podane dane mieszczą się w określonych powyżej zakresach, a w przypadku błędu pytać prosić użytkownika ponownie o podanie danych.

3. Napisać program, który pobiera od użytkownika liczbę całkowitą dodatnią, a następnie wyświetla na ekranie kolejno wszystkie liczby nieparzyste nie większe od podanej liczby. Przykład, dla 15 program powinien wyświetlić 1, 3, 5, 7, 9, 11, 13, 15.

4. Napisać program, który wczytuje od użytkownika liczbę całkowitą dodatnią n, a następnie wyświetla na ekranie wszystkie potęgi liczby 2 nie większe, niż podana liczba. Przykładowo, dla liczby 71 program powinien wyświetlić:

```
1
2
4
8
16
32
64
```

5. Napisać program, który wczytuje liczby podawane przez użytkownika dotąd, dopóki nie podana zostanie liczba 0. Następnie wyświetlić sumę wszystkich podanych liczb.

6. Napisać program, który pobiera od użytkownika ciąg liczb całkowitych. Pobieranie danych kończone jest podaniem wartości 0 (nie wliczana do danych). W następnej kolejności program powinien wyświetlić sumę największej oraz najmniejszej z podanych liczb oraz ich średnią arytmetyczną.

   Przykład:

   Użytkownik podał ciąg: 1, -4, 2, 17, 0.

   Wynik programu:

   ```
   13 // suma min. i maks.
   6.5 // średnia
   ```

7. Gra w "Za dużo, za mało". Komputer losuje liczbę z zakresu $1 \ldots 100$, a gracz (użytkownik) ma za zadanie odgadnąć, co to za liczba poprzez podawanie kolejnych wartości. Jeżeli podana wartość jest:

   - większa – wyświetlany jest komunikat „Podałeś za dużą wartość",
   - mniejsza – wyświetlany jest komunikat „Podałeś za małą wartość",
   - identyczna z wylosowaną – wyświetlany jest komunikat „Gratulacje" i gra się kończy.

8. Napisać program, który pobiera od użytkownika liczbę całkowitą, a następnie:

   - oblicza sumę cyfr tej liczby,
   - stosunek średniej arytmetycznej cyfr parzystych do średniej arytmetycznej cyfr nieparzystych.

9. Napisać program, dla podanej liczby całkowitej wyświetla jej dzielniki. Przykładowo, dla liczby 21 dzielniki to: 1, 3, 7, 21.

10. Napisać program, który sprawdza, czy podana liczba całkowita $n, n > 1$, jest liczbą pierwszą.

Zajęcia wstępne – listy

1. Napisać program, który:

   - utworzy listę 10 liczb całkowitych i wypełni ją wartościami losowymi z przedziału $[-10, \ldots, 10]$,
   - wypisze na ekranie zawartość listy,
   - wyznaczy najmniejszy oraz największy element w liście (za pomocą min / max oraz samemu),
   - wyznaczy średnią arytmetyczną elementów listy,
   - wyznaczy ile elementów jest mniejszych, ile większych od średniej.
   - wypisze na ekranie zawartość listy w odwrotnej kolejności, tj. od ostatniego do pierwszego.

   Wszystkie wyznaczone wartości powinny zostać wyświetlone na ekranie.

   ```
   Wylosowane liczby:
     -3    9    2  -10   -3   -4   -1   -5  -10    8
   Min: -10, max: 9
   Średnia: -1,00
   Mniejszych od śr.: 6
   Większych od śr.: 3
   Liczby w odwrotnej kolejności:
      8  -10   -5   -1   -4   -3  -10    2    9   -3
   ```

2. Napisać program, który utworzy listę 20 liczb całkowitych z przedziału $1 \ldots 10$, a następnie wypisze na ekranie ile razy każda z liczb z tego przedziału powtarza się w liście.

   Przykład:

   ```
   Wylosowane liczby: 6 5 4 5 10 5 8 3 10 6 6 6 4 3 2 8 1 3 4 7
   Wystąpienia:
    1 - 1
    2 - 1
    3 - 3
    4 - 3
   ```

```
 5 - 3
 6 - 4
 7 - 1
 8 - 2
 9 - 0
10 - 2
```

3. Napisz program, który:

   - stworzy listę list (macierz) 5 x 5 liczb całkowitych,
   - wypełnij ją losowymi wartościami z zakresu $\{-5, -4, \ldots, 5\}$,
   - dla każdej kolumny wyznacz minimum,
   - dla każdej kolumny wyznaczy maksimum.

   Program ma wyświetlać listę wypełnioną liczbami oraz tablice z minimami oraz maksymami.

4. Napisz program, który pobiera od użytkownika dodatnią liczbę naturalną n i tworzy listę o rozmiarze $n \times n$. Następnie program powinien wypełnić utworzoną listę, tak by $a[i][j] = +$ jeżeli liczby $(i + 1)$ oraz $(j + 1)$ są względnie pierwsze, tzn. nie mają wspólnych dzielników poza 1. W pozostałych przypadkach .. Tak utworzoną listę należy wypisać na ekranie. Przykład:

```
Podaj liczbę (> 0): 10
    1  2  3  4  5  6  7  8  9 10
 1  +  +  +  +  +  +  +  +  +  +
 2  +  .  +  .  +  .  +  .  +  .
 3  +  +  .  +  +  .  +  +  .  +
 4  +  .  +  .  +  .  +  .  +  .
 5  +  +  +  +  .  +  +  +  +  .
 6  +  .  .  .  +  .  +  .  .  .
 7  +  +  +  +  +  +  .  +  +  +
 8  +  .  +  .  +  .  +  .  +  .
 9  +  +  .  +  +  .  +  +  .  +
10  +  .  +  .  .  .  +  .  +  .
```

Zajęcia wstępne – łańcuchy znaków i funkcje

1. Napisać program, który wczytuje od użytkownika ciąg znaków, a następnie wyświetla informację o tym ile razy w tym ciągu powtarza się jego ostatni znak.

   Przykład, dla ciągu „Abrakadabra" program powinien wyświetlić 4, ponieważ ostatnim znakiem jest literka „a", która występuje w podanym ciągu łącznie 4 razy.

2. Napisać program, który wczytuje od użytkownika ciąg znaków, a następnie tworzy łańcuch będący odwróceniem podanego łańcucha i wyświetla go na ekranie. Przykładowo, dla łańcucha „Kot" wynikiem powinien być łańcuch „toK".

3. Napisać program, który wczytuje od użytkownika ciąg znaków, a następnie sprawdza, czy podany ciąg jest palindromem.

4. Napisać program, który sumuje cyfry w tekście podanym przez użytkownika.

   ```
   Przykład:
       "Ala ma 1 psa i 2 koty. Jola ma 10 rybek i 2 papugi."
   Wynik:
       6
   ```

5. Napisz program, który umożliwia szyfrowanie podanego ciągu znaków przy użyciu szyfru Cezara, który jest szczególnym przypadkiem szyfru podstawieniowego monoalfabetycznego.

   Użytkownik program podaje tekst do zaszyfrowania oraz liczbę n, o którą przesunięty jest alfabet za pomocą którego szyfrujemy tekst. Dla uproszczenia można przyjąć, że łańcuch wejściowy składa się tylko z małych liter alfabetu angielskiego, tj. 'a' – 'z' (26 znaków) oraz spacji.

   Przykład 1.

   ```
   Podaj łańcuch znaków do zaszyfrowania: abrakadabraz
   Podaj przesunięcie: 2
   Zaszyfrowany tekst: cdtcmcfcdtcb
   ```

   Przykład 2.

```
Podaj łańcuch znaków do zaszyfrowania: cdtcmcfcdtcb
Podaj przesunięcie: -2
Zaszyfrowany tekst: abrakadabraz
```

6. Zdefiniować funkcję `strToInt(str)`, która zamienia liczbę całkowitą zapisaną w postaci łańcucha na liczbę całkowitą typu int. Funkcja powinna przerywać konwersję w momencie napotkania pierwszego znaku nie należącego do zapisu liczby, zatem np. dla `strToInt("-13krowa")` wynikiem powinno być -13.

   Pozostałe przykłady:

   ```
   strToInt("+12") - wynik 12
   strToInt("0001") - wynik 1
   strToInt("991-234-23") - wynik 991
   strToInt("+zonk") - wynik 0
   strToInt("") - wynik 0
   strToInt("-12e5") - wynik -12*10^5 = -120000
   strToInt("-12e-5") - wynik -12, tylko dodatnie wykładniki są rozpatrywane
   ```

7. Zdefiniować funkcję `strfind(gdzie, co)`, która szuka łańcucha `co` w łańcuchu `gdzie` i jeżeli go znajdzie, to jej wynikiem jest pozycja, na której ten łańcuch zaczyna się w łańcuchu `gdzie`. Jeżeli nie udało się znaleźć łańcucha to wtedy wynikiem ma być -1.

   Przykłady:

   ```
   strfind("Ala ma kota", "ma") - wynik to 4
   strfind("Ala ma kota", "Ala ma kota") - wynik to 0
   strfind("Ala ma kota", "") - wynik to 0, bo pusty łańcuch jest
   podłańcuchem każdego innego łańcucha
   strfind("Pies", "jakis napis") - wynik to -1
   strfind("Ala ma kota", "pies") - wynik to -1
   ```

8. Zdefiniować funkcję `strFindAndCount(String gdzie, String co)`, która zlicza wystąpienia łańcucha `co` w łańcuchu `gdzie`. Jej wynikiem jest wyznaczona liczba wystąpień. Jeżeli nie udało się znaleźć łańcucha, to wtedy wynikiem jest, oczywiście, 0.

   Przykłady:

   ```
   strFindAndCount("Ala ma kota", "ma") - wynik to 1
   strFindAndCount("mama ma kota", "ma") - wynik to 3
   strFindAndCount("Ala mmaa ma kota", "ma") - wynik to 2
   strFindAndCount("Ala ma kota", "Asia") - wynik to 0
   ```

9. Napisać funkcję `czyAnagram(t1, t2)`, która sprawdza, czy łańcuch `t2` jest anagramem tekstu `t1`, czyli czy składa się z tych samych znaków, ale ustawionych niekoniecznie w tej samej kolejności

   Uwaga, należy sprawdzać jedynie małe i duże litery alfabetu angielskiego, jednak bez względu na ich wielkość, tzn. zarówno małe 'a' jak i duże 'A' liczone są tak samo. Pozostałe znaki nie są sprawdzane, a więc nie mają wpływu na to, czy słowo będzie uznane za anagram innego.

   Przykładowo, dla poniższego fragmentu programu:

   ```
   czyAnagram("kolej", "olejk") -> true
   czyAnagram("kolej", "kole") -> false
   czyAnagram("kolej", "K O L E J") -> true
   czyAnagram("Gregory House", "Huge ego, sorry") -> true
   ```

10. W języku HTML oraz kaskadowych arkuszach stylów (CSS) powszechne jest ustalanie kolorów elementów w postaci łańcucha `#RRGGBB`, gdzie RR jest dwucyfrową liczbą (od 0x0 do 0xFF) w kodzie szesnastkowym oznaczającą ile czerwieni jest w wynikowym kolorze. Analogicznie GG oznacza nasycenie zieleni, a BB niebieskiego.

    Napisać funkcję `HTMLColor2RGB(color)`, która jako parametr przyjmuje łańcuch postaci "#RRGGBB" i zwraca tablicę 3 liczb całkowitych w systemie 10 oznaczających nasycenie poszczególnych składowych.

    Przykład

    ```
    Wynikiem HTMLColor2RGB("#FF0050") powinna być tablica { 255, 0, 80 }.
    Wynikiem HTMLColor2RGB("#001020") powinna być tablica { 0, 16, 32 }.
    ```

Zadania wprowadzające – pliki tekstowe

1. Napisać funkcję liczZnakiSlowa, która zlicza:

   - liczbę znaków w pliku,
   - liczbę białych znaków w pliku (białe znaki to spacja, tabulator, znacznik końca wiersza),
   - liczbę słów w pliku.

   Wynikiem funkcji jest tablica złożona z 3 liczb całkowitych po jednej dla wymienionych podpunktów.

2. Napisać funkcję:

   ```
   def szukaj(nazwaPlikWe, nazwaPlikWy, slowo)
   ```

   której zadaniem jest znalezienie wszystkich wierszy w pliku, które *zawierają* szukane słowo. Wszystkie wiersze, które zawierają słowo powinny zostać zapisane w pliku wynikowym wraz z nr wiersza (z pierwszego pliku). Nazwa pierwszego pliku zapamiętana jest w parametrze `nazwaPlikWe`, nazwa pliku wynikowego podana jest w parametrze `nazwaPlikWy`, natomiast szukane słowo w parametrze `slowo`.

   Przykład - plik wejściowy:

   ```
   Ala ma jutro egzamin z biologii.
   Jan myje auto.
   Eh, jutro kolejny egzamin.
   Nie lubie polityki.
   ```

   Jeżeli szukanym słowem byłoby "egzamin", to plik wynikowy powinien wyglądać następująco:

   ```
   1: Ala ma jutro egzamin z biologii.
   3: Eh, jutro kolejny egzamin.
   ```

3. Napisać funkcję `def sumujIZapisz(nazwaPliku)`, która odczytuje plik o podanej nazwie zawierający liczby całkowite (po jednej w wierszu). Funkcja ma za zadanie odczytać i zsumować wszystkie liczby z pliku, a następnie dopisać na

końcu pliku wyznaczoną sumę powiększoną o 1. Ponowne uruchomienia funkcji będą skutkowały dopisywaniem kolejnych wierszy. Jeżeli plik nie istnieje to ma zostać utworzony – suma dla pustego pliku wyniesie 0, a więc należy dopisać wiersz zawierający 1.

4. Stworzyć dwie funkcje:

```
def szyfruj(nazwaWe, przesun)
def deszyfruj(nazwaWe, przesun)
```

   Funkcja szyfruj dokonuje szyfrowania pliku, którego nazwa podana została jako pierwszy parametr. Szyfrowanie polega na zamianie każdej litery na znak przesunięty o wartość podaną drugim parametrem np. dla przesunięcia równego 2 literka 'a' powinna zostać zastąpiona literką 'c', literka 'z' literką 'b' itp.

   Wynikiem działania funkcji ma być plik o nazwie utworzonej na podstawie nazwy pliku wejściowego poprzez dołączenie znaku '_' np. dla pliku dane.txt zaszyfrowana postać powinna mieć nazwę _dane.txt. Funkcja deszyfruj powinna deszyfrować plik (niekoniecznie ten sam) zaszyfrowany przez funkcję szyfruj.

5. Napisać funkcję def emerytura(nazwaPliku), która wczyta z pliku o podanej nazwie dane pracowników zapisane w kolejnych wierszach w następujący sposób:
   Imię Nazwisko Płeć Wiek

```
Przykład:
Tomasz Nowak M 45
Marta Ziobro K 42
Jan Kowalski M 27
Ewelina Tusk K 59
```

   Następnie funkcja dla każdego pracownika powinna wyznaczyć ile lat pozostało do jego emerytury. Wyniki należy zapisać w następujący sposób:
   Nazwisko Imię "Lata do emerytury"

```
Przykład:
Nowak Tomasz 20
Kowalski Jan 38
```

   Wyniki dla kobiet należy zapisać w pliku o nazwie "kobiety.txt", natomiast wyniki dla mężczyzn należy zapisać w pliku "mezczyzni.txt".

6. Napisać funkcję, której zadaniem jest odczytanie danych tabelarycznych z pliku tekstowego, a następnie zapisanie ich do nowego pliku w postaci kodu HTML.

```
Przykład:
Wejście:
"Waga" "Wzrost" "BMI" "Nadwaga"
70 1,8 21,6 "NIE"
67 1,77 21,39 "NIE"
85 1,7 29,41 "TAK"
100 1,92 27,13 "TAK"
Wynik:
<html><body>
<table>
<tr><td>"Waga"</td><td>"Wzrost"</td><td>"BMI"</td><td>"Nadwaga"
</td></tr>
<tr><td>70</td><td>1,8</td><td>21,6</td><td>"NIE"
</td></tr>
<tr><td>67</td><td>1,77</td><td>21,39</td><td>"NIE"
</td></tr>
<tr><td>85</td><td>1,7</td><td>29,41</td><td>"TAK"
</td></tr>
<tr><td>100</td><td>1,92</td><td>27,13</td><td>"TAK"</td></tr>
</table>
</body></html>
```

7. Napisać program, który dla pliku tekstowego o podanej nazwie wyznaczy „wykres" częstości wystąpień małych liter alfabetu angielskiego. Słupki wykresu mają zostać utworzone ze znaków gwiazdki '*', przy czym długość słupka dla najczęściej występującej litery powinna wynosić 50. Dodatkowo dla każdego znaku należy dodatkowo wyświetlić liczbę jego wystąpień.

   Poniżej umieszczono przykładowy wykres wygenerowany dla tekstu „Adventures of Huckleberry Finn" M. Twaina dostępnego pod adresem:
   `http://www.gutenberg.org/dirs/7/76/76.txt`

```
a ************************************          36581
b *******                                     7439
c ********                                    8317
d ************************                    23754
e ************************************************** 49084
f ********                                    7914
g **********                                  10733
h *************************                   26338
i ****************************                28222
j *                                           1211
k *****                                       5677
l ****************                            17446
m **********                                  10337
```

```
n ********************************      32818
o ************************************  36700
p ******                                5971
q                                       189
r ********************                  20252
s ************************              25193
t ****************************************** 42390
u **************                        13954
v **                                    2944
w *************                         13347
x                                       453
y **********                            10312
z                                       185
```

Zadania z podstaw programowania obiektowego

1. Napisać klasę Wektor reprezentującą wektor na płaszczyźnie kartezjańskiej. Klasa ta powinna mieć dwie składowe prywatne – współrzędne (x, y) oraz następujące metody publiczne:

   - konstruktor z parametrami umożliwiającymi ustalenie wartości współrzędnych wektora;
   - wartości domyślne do konstruktora pozwalające na stworzenie wektora zerowego;
   - metodę `dlugosc(self)`, której wynikiem ma być długość wektora;
   - metodę `normalizuj(self)`, której wynikiem powinien być nowy wektor równy znormalizowanemu bieżącemu wektorowi;
   - metodę `wyswietl(self)`, której wynikiem powinien być łańcuch opisujący wektor oraz jego długość, np. "Wektor [1.5, 0] o długości 1.5".

   Przeciążyć:

   - operator dodawania `__add__(self, w)`, którego wynikiem ma być nowy wektor będący sumą bieżącego i podanego wektora;
   - operator odejmowania `__sub__(self, w)`, którego wynikiem ma być nowy wektor będący różnicą bieżącego i podanego wektora;
   - analogicznie (tym razem modyfikując `self`) operatory `+=` oraz `-=`, czyli odpowiednio `__iadd__` oraz `__isub__`;
   - wbudowaną funkcję `str()`, czyli `__str(self)__`, aby wynikiem był łańcuch opisujący wektor, np. "[1.2, -3]";
   - operator mnożenia `__mul__(self,a)`, której wynikiem ma być nowy wektor będący wynikiem przemnożenia bieżącego wektora przez podany skalar `a`;
   - analogicznie `__rmul__(self,a)`, dla wywołania `a*wektor`.

   Przykład zastosowania klasy Wektor:

```
w1 = Wektor(2,4)
w2 = Wektor(1.5)
print("Wektor w1:", w1, "wektor w2:", w2)
print("Dł. w1 =", w1.dlugosc(), "dł. w2 =", w2.dlugosc())
print("w1+w2 =", w1+w2)
```

```
print("w1-w2 =", w1-w2)
print("w1*2 =", w1*2)
print("-3*w2 =", -3*w2)
print("w1*2-w2 =", w1*2-w2)
print("w1 po normalizacji =", w1.normalizuj())
print("w2 po normalizacji =", w2.normalizuj())
w1.wyswietl()
w2.wyswietl()
```

Wynik działania przykładowego programu:

```
Wektor w1: [2.00, 4.00] wektor w2: [1.50, 0.00]
Dł. w1 = 4.47213595499958 dł. w2 = 1.5
w1+w2 = [3.50, 4.00]
w1-w2 = [0.50, 4.00]
w1*2 = [4.00, 8.00]
-3*w2 = [-4.50, 0.00]
w1*2-w2 = [2.50, 8.00]
w1 po normalizacji = [0.45, 0.89]
w2 po normalizacji = [1.00, 0.00]
Wektor [2.00, 4.00] o długości 4.47213595499958
Wektor [1.50, 0.00] o długości 1.5
```

2. Napisać program do obsługi zamówień. W tym celu należy zaimplementować klasy ElementZamowienia oraz Zamowienie.

   Klasa ElementZamowienia reprezentuje pojedynczą pozycję zamówienia i zawiera prywatne pola:

   - `nazwa`,
   - `cena`,
   - `liczbaSztuk`.

   Dodatkowo klasa ta powinna składać się z:

   - konstruktora z parametrami umożliwiającymi przypisanie wartości początkowych polom klasy;
   - przeciążonej wbudowanej funkcji `str()`, aby wynikiem był łańcuch łańcuch znaków opisujący element zamówienia, np. "Chleb 4.00 zł, 2 szt., łącznie 8.00 zł";
   - metody `obliczKoszt` (bez dodatkowych parametrów), służącej do zwracania łącznego kosztu danej pozycji zamówienia z uwzględnieniem rabatu;
   - metody `obliczRabat` (bez dodatkowych parametrów), służącej do wyznaczania rabatu dla klienta, według zasad: jeżeli klient zamówił co najmniej 5 sztuk tego towaru to rabat wynosi 10%, w przeciwnym razie 0%.

   Klasa Zamowienie zawiera listę zamówień i następujące pola prywatne:

- `elementy` – lista z elementami zamówienia (typu `ElementZamowienia`),
- `rozmiar` – aktualna liczba elementów w zamówieniu,
- `maksRozmiar` – maks. liczba elementów w zamówieniu.

Dodatkowo klasa ta powinna składać się z:

- konstruktora z jednym parametrem określającym maks. liczbę elementów zamówienia;
- metody `dodaj(self, elZam)` służącej do dodania nowego elementu zamówienia do zamówienia. Metoda zwraca `True`, jeśli udało się dodać element. W przypadku przekroczenie wielkości zamówienia zwraca `False` i dodając nic do zamówienial;
- metody `obliczKoszt(self)` służącej do obliczenia całkowitego koszt zamówienia (zwraca tę wartość);
- metody `pisz(self)` służącej do wypisywania na ekranie informacji o zamówieniu tj. listy pozycji, łączny koszt oraz naliczony rabat.

Przykład zastosowania klas:

```
z = Zamowienie(10)
z.dodaj(ElementZamowienia("Chleb", 4.0, 2))
z.dodaj(ElementZamowienia("Mleko", 2.5, 1))
z.dodaj(ElementZamowienia("Cukier", 4.0, 5))
z.dodaj(ElementZamowienia("Puszka", 9.0, 1))
z.pisz()
```

Przykładowy wydruk:

```
Zamowienie:
    1. Chleb 4.00 zł, 2 szt., łącznie 8.00 zł
    2. Mleko 2.50 zł, 1 szt., łącznie 2.50 zł
    3. Cukier 4.00 zł, 5 szt., łącznie 18.00 zł
    4. Puszka 9.00 zł, 1 szt., łącznie 9.00 zł
Koszt całkowity: 37.5 zł
Naliczony rabat: 2 zł
```

Zadania z programowania obiektowego

1. Napisać klasę `Samochod` zawierającą pola `marka` i `zuzyciePaliwa` oraz następujące metody publiczne:

   - konstruktor z parametrami umożliwiającymi ustalenie wartości pół (uwzględnić też wartości domyślne);
   - metodę `koszt(self, km, cena)`, zwracającą koszt paliwa dla danych z parametru,;
   - metodę `wyswietl(self)`, której wynikiem powinien być łańcuch opisujący samochód (podanie marki i zużycia paliwa).

   Zdefiniować klasy pochodne:

   - `Autobus`, w której dodatkowo powinno się znajdować pole `lMiejsc`;
   - `Ciezarowka`, w której dodatkowo powinno się znajdować pole `nosnosc`.

   Klasy pochodne powinny zawierać odpowiednie konstruktory (analogicznie do klasy bazowej) oraz metody wypisujące dane na konsolę `wyswietl(self)`.

   Stworzyć obiekt klasy `Samochod`, `Autobus` i `Ciezarowka` oraz zastosować metody `koszt` (wyświetlić rezultat) oraz `wyswietl`.

2. Napisać klasę `Statek` (do popularnej gry w statki) zawierającą pola `wspolrzedne1` `wspolrzedne2` i `maszt` (wartość logiczna – `False` oznacza, że maszt został trafiony) oraz następujące metody publiczne:

   - konstruktor z parametrami umożliwiającymi ustalenie wartości pół współrzędnych (maszt należy ustawić na `True`);
   - metodę `strzal(self, wsp1, wsp2)`, sprawdzającą, czy wykonany strzał trafia w maszt. Metoda zwraca 0 – jeśli strzał jest chybiony, 1 – jeśli statek jest trafiony i 2 – jeśli jest trafiony i zatopiony,
   - metodę `wyswietl(self)`, wypisującą na konsolę dane (współrzędne).

   Zdefiniować klasy pochodne klasy `Statek`:

   - `Statek2Maszty`, w której dodatkowo powinno się znajdować pole `maszt2`;
   - `Statek3Maszty`, w której dodatkowo powinny się znajdować pola `maszt2` i `maszt3`.

Klasy pochodne powinny zawierać odpowiednie konstruktory (analogicznie do klasy bazowej) oraz metody realizujące strzał (`strzal(self, wsp1, wsp2)`) – zakładamy, że każdy statek ma tylko pojedyncze współrzędne, a kolejne maszty stawiane są zawsze poziomo, w prawą stronę (np. dla statku 3 masztowego o współrzędnych B3 maszty są na B3, B4 i B5). Zastosować pełną hermetyzację.

Stworzyć obiekt klasy `Statek`, `Statek2Maszty` i `Statek3Maszty` oraz zastosować metody `strzal` (wyświetlić rezultat) oraz `wyswietl`.