Laboratorium grafowe 1

Projekt „Matematyka dla Ciekawych Świata”,
Tomasz Świerczewski, Jakub Jałowiec, Łukasz Mazurek

15.03.2021

1 Praca z Pythonem

Na zajęciach będziemy programować w języku Python w wersji 3. Pythona można używać na jeden z dwóch sposobów:

Zainstalowany interpreter Pythona. Interpreter Pythona może być zainstalowany na komputerze, z którego korzystamy. Można go bezpłatnie pobrać ze strony https://www.python.org/downloads/ i zainstalować na swoich domowych komputerach. Zwróćcie uwagę, aby zainstalować wersję o numerze rozpoczynającym się od 3 (np. 3.x), a nie starszą, ale wciąż używaną wersję 2. Wersje te różnią się tak znacząco, że programy, które będziemy pisać na zajęciach nie będą działać w starszej, drugiej wersji Pythona.

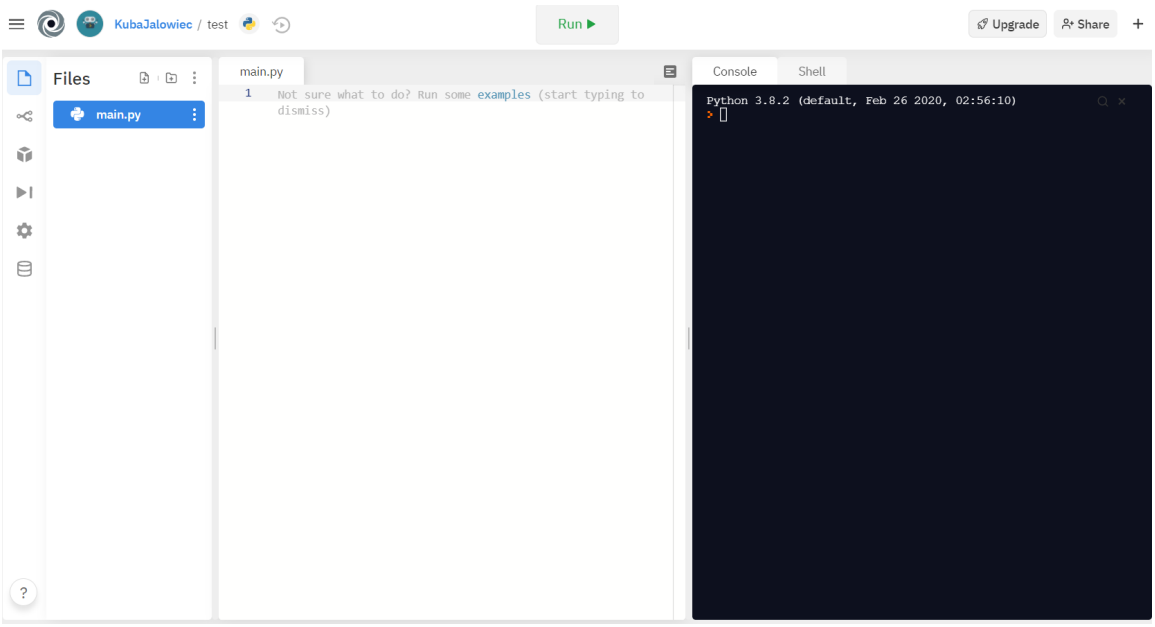
Interpreter Pythona online. Istnieje wiele interpreterów Pythona online. Na zajęciach będziemy korzystali z interpretera znajdującego się na stronie http://repl.it. Interpretera tego będziecie mogli również używać na swoich domowych komputerach bez instalacji żadnego dodatkowego oprogramowania — wystarczy dowolna przeglądarka i podłączenie do internetu.

2 Praca z interaktywną konsolą Pythona `repl.it`

2.1 Wprowadzenie. Podstawowe operacje.

Jeśli nie posiadasz jeszcze konta na http://repl.it to musisz je założyć. W ramach pomocy możesz posiłkować się utworzonym do tego skryptem.

Jeśli posiadasz już konto, to zaloguj się do niego, utwórz nowy projekt lub otwórz istniejący. Zobaczysz dwa okienka: miejsce na kod (białe, z lewej) i *konsolę* (czarne, z prawej, zwane również *terminalem*), tak jak na rysunku 1.



Rysunek 1: Widok interpretera Pythona dostępnego na stronie http://repl.it

Pierwszym sposobem pracy z Pythonem jest praca w interaktywnej konsoli, czyli praca w prawym okienku. W konsoli tej początkowo wypisane są pewne informacje (m. in. używana wersja Pythona) oraz znak zachęty >. Interpreter oczekuje, iż po tym znaku wpiszemy polecenie i naciśniemy Enter. Wynik polecenia zostanie wypisany w kolejnym wierszu (poprzedzony znakiem =>). Najprostszym sposobem użycia konsoli Pythona jest użycie jej jako kalkulatora — wpisujemy działanie do obliczenia, naciskamy Enter i w kolejnym wierszu otrzymujemy wynik działania. Przykład użycia konsoli Pythona jako kalkulatora znajduje się poniżej:

```
 > 2 + 2 * 2
=> 6
 > (2 + 2) * 2
=> 8
 > 2 ** 7
=> 128
 > 47 / 10
=> 4.7
 > 47 // 10
=> 4
 > 47 % 10
=> 7
```

W powyższym przykładzie:

- Znak ** oznacza podnoszenie do potęgi.
- Znak / oznacza dzielenie.
- Znak // oznacza dzielenie całkowite.
- Znak % oznacza branie reszty z dzielenia.

Podobnie jak w kalkulatorze możemy korzystać z *pamięci*, w Pythonie możemy zapisywać wartości w *zmiennych*:

```
 > x = 3
 > y = 4
 > x
=> 3
 > x**2 + y**2
=> 25
```

W pierwszych dwóch linijkach następuje *przypisanie* wartości 3 do zmiennej x oraz wartości 4 do zmiennej y. Od tej pory możemy korzystać z tych zmiennych, np. do obliczenia wartości wyrażenia $(x^2 + y^2)$.

Zadanie 2.1.1

Korzystając z Pythona jak z kalkulatora, znajdź wszystkie dziesięciocyfrowe potęgi dwójki.

Zadanie 2.1.2

Spróbuj przy pomocy wcześniej poznanych operatorów zaokrąglić w dół następujące liczby: `10.7`, `1.1234`, `3.5`.

2.2 Pętla `for`

Załóżmy, że chcemy obliczyć kwadraty wszystkich liczb od 1 do 5. Zgodnie z dotychczasową wiedzą, w tym celu musimy wykonać 5 działań:

```
 > 1 * 1
=> 1
```

```
 > 2 * 2
=> 4
 > 3 * 3
=> 9
 > 4 * 4
=> 16
 > 5 * 5
=> 25
```

Widzimy jednak, że te działania są bardzo podobne i chciałoby się je wykonać „za jednym zamachem”. Do wykonywania wielokrotnie tego samego (lub podobnego) kodu służą pętle. Najprostszym rodzajem pętli jest pętla **for**, która dla danej *listy* i operacji do wykonania wykonuje tę operację po kolei na każdym elemencie listy.

Do wykonania powyższego zadania służy pętla **for** w następującej postaci:

```
 > for x in [1, 2, 3, 4, 5]:
..     print(x * x)
..
1
4
9
16
25
```

Spróbuj przepisać tę pętlę do konsoli interpretera repl.it i uruchomić. Zwróć uwagę na kilka rzeczy:

- Na końcu pierwszej linijki jest dwukropek.
- Gdy naciśniemy Enter po zakończeniu pierwszej linijki, znak zachęty zmieni się z > na . . , co oznacza to, że jesteśmy w trakcie pisania polecenia wielolinijkowego.
- Druga linijka musi być *wcięta*, tzn. rozpoczynać się od spacji, kilku spacji lub znaku tabulacji. Jeśli zapomnimy o wcięciu, interpreter zgłosi błąd i całą komendę wielolinijkową będziemy musieli pisać od początku (można pomóc sobie, naciskając strzałkę w górę i przeglądając stare komendy).
- Po wpisaniu drugiej linijki i naciśnięciu Entera pojawi się kolejny znak zachęty . . , co oznacza, że interpreter czeka na ciąg dalszy polecenia wielolinijkowego. Jeśli nie chcemy pisać dalszego ciągu, w tym momencie musimy jeszcze raz nacisnąć Enter.

3 Pisanie i uruchamianie kodu programu

Do tej pory korzystaliśmy z Pythona używając interaktywnej konsoli (czyli prawego okienka). Jest to całkiem wygodne narzędzie, jeśli wykonujemy tylko jednolinijkowe polecenia, jednak pisanie dłuższych fragmentów kodu w tej konsoli staje się już bardzo niewygodne. Drugą metodą korzystania z Pythona jest pisanie kodu programu (skryptu) w pliku tekstowym (w lewym okienku) i uruchamianie tego kodu w konsoli (w prawym okienku).

Spróbuj teraz napisać w lewym okienku kod tej samej pętli for (pamiętając o dwukropku na końcu pierwszej i wcięciu drugiej linijki):

```
for x in [1, 2, 3, 4, 5]:
    print(x * x)
```

i wcisnąć przycisk „run” znajdujący się na górze tego okienka. Po chwili w konsoli po prawej powinien pojawić się wynik działania naszego programu:

```
1
4
9
16
25
```

Od tej pory właśnie w taki sposób będziemy pisać i uruchamiać wszystkie nasze programy. Ilekroć w niniejszych materiałach pojawią się dwie ramki, jedna obok drugiej, w lewej ramce znajdował się będzie kod programu, a w prawej efekt jego działania wyświetlony w konsoli:

```
1 for x in [1, 2, 3, 4, 5]:
2     print(x * x)
```

```
1 1
2 4
3 9
4 16
5 25
```

Powyższa pętla wypisała każdą liczbę w osobnej linijce. Dzieje się tak, ponieważ funkcja `print(...)` domyślnie przechodzi do następnej linii po każdym wywołaniu. Można jednak zmienić to zachowanie, dodając wewnątrz `print(...)` po przecinku końcówkę `end = X`, gdzie `X` to otoczony apostrofami ciąg znaków, który chcemy wypisywać zamiast przejścia do nowej linii.

Przykłady:

```
1 for x in [1, 2, 3, 4, 5]:
2     print(x * x, end = ' ')
```

```
1 1 4 9 16 25
```

```
1 for x in [1, 2, 3, 4, 5]:
2     print(x * x, end = '')
```

```
1 1491625
```

```
1 for x in [1, 2, 3, 4, 5]:
2     print(x * x, end = ' , ')
```

```
1 1 , 4 , 9 , 16 , 25 ,
```

4 Listy

4.1 Lista kolejnych liczb naturalnych

Często potrzebujemy, aby pętla przeszła po liście kilku kolejnych liczb naturalnych. W tym celu możemy oczywiście podać wprost kolejne elementy listy (tak jak w powyższym przykładzie), jednak istnieje wygodniejsze rozwiązanie, mianowicie polecenie `range()`:

```
1 for x in range(7):
2     print(x, end = ', ')
```

```
1 0, 1, 2, 3, 4, 5, 6,
```

```
1 for x in range(5, 10):
2     print(x, end = ', ')
```

```
1 5, 6, 7, 8, 9,
```

```
1 for x in range(10, 20, 3):
2     print(x, end = ', ')
```

```
1 10, 13, 16, 19,
```

Na powyższych przykładach widzimy, że polecenie `range()` występuje w trzech wersjach:

- `range(kon)` generuje listę kolejnych liczb od 0 (**włącznie**) do `kon` (**wyłącznie**).
- `range(pocz, kon)` generuje listę kolejnych liczb od `pocz` (**włącznie**) do `kon` (**wyłącznie**).
- `range(pocz, kon, krok)` generuje listę liczb od `pocz` (**włącznie**) do `kon` (**wyłącznie**), przeskakując w każdym kroku o `krok`.

Do zapamiętania: ***Wszystkie przedziały w Pythonie są domknięte z lewej strony i otwarte z prawej strony, tzn. zawierają swój lewy koniec i nie zawierają swojego prawego końca.***

Zadanie 4.1.1

Korzystając z Pythona wypisz na konsolę nieparzyste liczby niezerowe, których moduł jest mniejszy niż 20, w kolejności od liczb z najmniejszym modułem, do liczb z największym modułem, tzn.:

```
-1
1
-3
3
-5
5
itd.
```

4.2 Słowa

Do tej pory używaliśmy zmiennych do przechowywania liczb i operowania na nich:

```
 > a = 2
 > b = 5
 > a + b
=> 7
 > a**b
=> 32
```

Zmienne mogą również jako wartości przyjmować litery, słowa, a nawet całe zdania:

```
 > x = 'A'
 > a = 'Ala'
 > b = "ma"
 > c = 'kota i wiele innych zwierząt'
 > x
=> 'A'
 > a[0]
=> 'A'
 > c[-1]
=> 't'
 > c[1]
=> 'o'
 > a + b
=> 'Alama'
 > a + b + c
=> 'Alamakota i wiele innych zwierząt'
```

Zwróć uwagę na następujące rzeczy:

- Napisy muszą być otoczone pojedynczymi apostrofami lub podwójnym cudzysłowami (nie ma znaczenia, którą wersję wybierzemy).
- Przy użyciu liczby w nawiasie kwadratowym możemy poznać poszczególne litery słowa (numeracja rozpoczyna się od 0).
- Przy użyciu znaku dodawania możemy sklejać (*konkatenować*) napisy.
- Jeśli użyjemy liczby -1 w nawiasie kwadratowym to odczytamy ostatnią literę.

Innymi przydatnymi operacjami na słowach jest sprawdzanie długości słowa poleceniem `len()`:

```
 > slowo = 'Python'
 > len(slowo)
=> 6
```

Zadanie 4.2.1

Napisz pętlę, która dla danego słowa wypisze najpierw jego pierwszą literę, następnie ostatnią, drugą, pierwszą, trzecią, drugą, czwartą, trzecią itd. Czyli zamiast każdej litery w słowie wypisze tę literę, a później tę, która jest przed nią. Dla napisu `'Napis'` chcemy uzyskać `'N, s, a, N, p, a, i, p, s, i,'`.

Zadanie 4.2.2

Napisz pętlę, która dla danego słowa wypisze w pierwszej linijce tylko jego pierwszą literę, w drugiej dwie pierwsze litery, w trzecim trzy itd.
Przykład:

```
M
Ma
Mat
Mate
Matem
Matema
Matemat
Matematy
Matematyk
Matematyka
```

4.3 Słowo jako lista liter

Wszystkie listy, których do tej pory używaliśmy w pętli **for** były listami liczb. Okazuje się, że w Pythonie pojedyncze słowo również jest listą, a dokładniej listą liter. Oznacza to, że po słowie można przejść przy użyciu pętli **for**, tak samo jak przechodziliśmy po liście liczb:

```
1 for litera in 'Kula':
2     print('litera' + ' ' + litera)
```

```
1 litera K
2 litera u
3 litera l
4 litera a
```

Zwróć uwagę, że wewnątrz pętli for może znajdować się więcej niż jedno polecenie. Trzeba tylko pamiętać, aby wszystkie były poprzedzone takim samym wcięciem.

4.4 Lista słów

Poznaliśmy już dwa rodzaje list: listę liczb i listę liter (czyli słowo). Okazuje się, że w Pythonie możemy tworzyć listy obiektów dowolnego rodzaju, np. listy słów.

```
lista = ['Ala', 'ma', 'kota']
```

Podobnie jak w przypadku listy liczb i listy liter, po liście słów również możemy przejść pętlą for:

```
for slowo in lista:
    print(slowo)
```

```
Ala
ma
kota
```

W każdym kroku takiej pętli kolejne słowo z listy jest zapisywane do zmiennej `slowo`, a następnie zawartość tej zmiennej jest wypisywana na ekran. Oczywiście przed wypisaniem można dowolnie zmodyfikować zmienną `slowo`:

```
for slowo in lista:
    nowe_slowo = slowo + 'hh' + slowo + 'hh'
    print(nowe_slowo, end = ' ')
```

```
AlahhAlahh mahhmahh kotahhkotahh
```

Na powyższym przykładzie widzimy, że:

- W kolejnych krokach pętli zmienna `slowo` zawiera kolejne słowa z listy.
- W każdym kroku pętli `slowo` jest sklejane ze słowem `'hh'`, a następnie mnożone przez 2 (czyli sklejane z samym sobą).
- Używając dodatkowego argumentu `end = ...` wymuszamy wypisanie spacji zamiast przejścia do nowej linii po końcu wypisywanego wyrażenia.

Skoro każde słowo to lista liter, to jak można uzyskać dostęp do liter na liście słów? Z pomocą przychodzą zagnieżdżone pętle for, takie jak ta poniżej:

```
for slowo in lista:
    for litera in slowo:
        print(litera, end = ' ')
```

```
A l a m a k o t a
```

Zadanie 4.4.1

Napisz pętlę, która dla danej listy słów `lista` utworzy akronim składający się z pierwszych liter każdego słowa. Np. dla listy `['Matematyka', 'Informatyka', 'Mechanika']` wypisze słowo `MIM`.

5 Instrukcja warunkowa `if`

Często chcemy, aby program zachowywał się w różny sposób w zależności od tego, czy jakiś warunek jest spełniony, czy nie. W Pythonie (jak w większości języków programowania) służy do tego instrukcja warunkowa `if`.

Przypuśćmy, że chcemy sprawdzić, czy `liczba` jest parzysta, czy nie i wypisuje odpowiedni komunikat. Aby sprawdzić, czy liczba jest parzysta, możemy sprawdzić, czy reszta z dzielenia jej przez 2 jest równa 0:

```
liczba = 3

if liczba % 2 == 0:
  print("Parzysta")
```

```
else:
  print("Nieparzysta")
```

Zwróć uwagę na następujące rzeczy:

- **if** to po polsku „jeśli”, **else** to po polsku „w przeciwnym przypadku”.
- Linijki rozpoczynające się od **if** i **else** (podobnie jak linijki rozpoczynające się od **for** lub **def**) kończą się dwukropkiem.
- „Wnętrze” **if**-a i **else**-a (linijki 4 i 6) jest wcięte (podobnie jak wnętrze **for**-a lub **def**-a).
- Linijka 4 zostanie wykonana, jeśli spełniony będzie warunek z linijki 3, czyli jeśli reszta z dzielenia liczby przez 2 będzie równa 0.
- Linijka 6 zostanie wykonana, jeśli warunek z linijki 3 nie będzie spełniony, czyli jeśli reszta z dzielenia liczby przez 2 nie będzie równa 0.

W powyższym przykładzie użyliśmy konstrukcji **if**/**else** do rozróżnienia pomiędzy dwoma przypadkami. Używając komendy **elif** (skrót od **else if**) możemy stworzyć bardziej skomplikowany kod do rozróżnienia pomiędzy kilkoma różnymi przypadkami:

```
1  for c in 'Analfabetyzm':
2    if c == 'A' or c == 'a':
3      print('A lub a')
4    elif c in 'xyz':
5      print('x-z')
6    else:
7      print('Nic ciekawego')
```

```
1  A lub a
2  Nic ciekawego
3  A lub a
4  Nic ciekawego
5  Nic ciekawego
6  A lub a
7  Nic ciekawego
8  Nic ciekawego
9  Nic ciekawego
10 x-z
11 x-z
12 Nic ciekawego
```

Ten kod składa się z trzech bloków, które są wykonywane w zależności od spełnienia poszczególnych warunków: **if**, **elif**, **else**. Mamy dużą dowolność w konstruowaniu tego typu fragmentów kodu: bloków **elif** może być dowolnie wiele, blok **else** może występować jako ostatni blok, ale może też go nie być w ogóle. W powyższym przykładzie widzimy również, jak można łączyć warunki spójnikiem **and** („i”) oraz **or** („lub”) Pierwsze połączenie oznacza, że chcemy, aby spełnione były **wszystkie** z wymienionych warunków, a drugie, że chcemy aby spełniony był **co najmniej jeden** z wymienionych warunków.

Zadanie 5.0.1

Sprawdź, czy punkty o współrzędnych równych: $p_x^1 = 10$ $p_y^1 = 2$, $p_x^2 = 10$ $p_y^2 = 5$, $p_x^3 = 8$ $p_y^3 = 8$, są wewnątrz okręgu, który ma środek w $o_x = 5$ $o_y = 5$ i promień r=5. Jeśli punkt jest wewnątrz wypisz "Wewnątrz", jeżeli jest na okręgu to wypisz "Na okręgu", a w przeciwnym przypadku "Na zewnątrz".

6 Zadania dodatkowe

Zadanie 6.0.1

Oblicz sumę $1^2 + 2^2 + 3^2 + ... + 99^2 + 100^2$.

Zadanie 6.0.2

Używając dwóch pętli `for`, jedna wewnątrz drugiej, napisz program, który wypisze na ekranie *trójkąt z siódemek*, taki jak poniżej:

```
7
77
777
7777
77777
777777
7777777
77777777
777777777
```

Zadanie 6.0.3

Napisz pętlę, która przejdzie po słowie zapisanym w zmiennej `slowo` wypisze je powtarzając każdą literę dwukrotnie. Np. dla `slowo = 'kalafior'` pętla powinna wypisać słowo `'kkaallaaffiioorr'`.

Zadanie 6.0.4* — zadanie z gwiazdką

To zadanie jest przykładem zadania z życia wziętego, rzeczywistym problemem matematycznym.

Korzystając z Pythona jak z kalkulatora oblicz próg rocznych zarobków dla osoby fizycznej prowadzącej działalność gospodarczą, dla których bardziej opłacalne jest z korzystanie podatku liniowego 19%, niż ze skali podatkowej, czyli 17% lub 32%.
Wystarczy dokładność do 100 złotych kwoty wysokości podstawy obliczania podatku.

Wskazówki:
W Polsce jest możliwość wybrania jednego z dwóch sposobów rozliczania się. Pierwszy z nich to podatek liniowy, który wynosi 19% niezależnie od wysokości podstawy obliczania podatku w złotych. Jeżeli wysokość podstawy obliczania podatku to np. 50000 złotych, to podatek wyniesie 9500 złotych, a dla dochodów równych 100000 złotych wyniesie 19000 złotych.

Drugim sposobem jest zastosowanie skali podatkowej. Dla dochodów wynoszących do 85528 złotych podatek wynosi 17%. Powyżej tej kwoty należy zastosować drugi próg podatkowy, a więc należy zapłacić podatek 17% z 85528 złotych i 32% z kwoty, która przekracza 85528 złotych. Dla 50000 złotych podatek wyniesie 8500 złotych, a dla dochodów równych 100000 złotych wyniesie:
$85528 \cdot 17\% = 14539,76$ plus $(100000 - 85528) \cdot 32\% = 14472 \cdot 32\% = 4631,04$,
co razem da 19170,80 złotych.

Niestety, aby utrudnić obliczenia, dla osoby fizycznej korzystającej ze skali podatkowej jest możliwość zastosowania kwoty zmniejszającej podatek odliczanej w rocznym obliczaniu podatku lub zeznaniu. Nie dotyczy ona do podatku obliczanego z podatku liniowego.

Dla dochodów mniejszych niż 8000 złotych mamy kwotę wolną od podatku, czyli jesteśmy zwolnieni z podatku, tzn. kwota zmniejszająca podatek wynosi $8000 \cdot 17\% = 1360$ złotych. Dla podstawy obliczeniowej podatku między 8000 a 13000 złotych stosuje się wzór:

$$1360 - (834,88 \cdot (\text{podstawa obliczania podatku} - 8000)/5000) \text{ złotych.}$$

Dla zakresu od 13000 do 85528 złotych stosuje się stałą kwotę zmniejszającą podatek, która wynosi 525,12 złotych. Dla zakresu od 85528 do 127000 złotych stosuje się wzór:

$$525,12 - (525,12 \cdot (\text{podstawa obliczenia podatku} - 85528)/41472) \text{ złotych.}$$

Dla podstawy obliczania podatku większej niż 127000 złotych w skali roku nie stosuje się kwoty zmniejszającej podatek.

Wracając do naszych przykładów, dla 50000 złotych mieliśmy podatek 8500 złotych, lecz pomniejszamy go o 525,12 złotych, więc uzyskujemy 7987,88 złotych podatku dochodowego. Dla 100000 złotych mieliśmy podatek 19170,90 złotych. Tym razem pomniejszamy go o:

$$525,12 - (525,12 \cdot (100000 - 85528)/41472) = 341,875 \approx 341,88 \text{ złotych,}$$

więc uzyskujemy 18829,02 złotych, co jest nadal o 170,98 złotych mniejszym podatkiem, niż przy zastosowaniu podatku liniowego.

Wskazówka 2:
Możesz w jednej zmiennej obliczyć dla danej wartości `x` wysokość podatku za pomocą skali podatkowej, a w drugiej przechowywać obliczaną z podatku liniowego. Następnie możesz wypisać obie te wartości na konsolę i dotąd zmieniać wartość `x`, aż ten podatek się zrówna.

7 Praca domowa nr 1

Rozwiązania zadań domowych należy przesłać do poniedziałku 22 marca do godz. 14^{59} na adres licealisci.pracownia@icm.edu.pl wpisując jako temat wiadomości `Lx PD1`, gdzie `x` to numer grupy, np. `L3 PD1` dla grupy L3, itd.

Rozwiązania można również wysłać w prywatnej wiadomości prowadzącemu zajęć na Discordzie. Prowadzący może również zmienić ostateczny termin przesłania pracy domowej, np. na inną godzinę lub dzień.

Za pracę domową można maksymalnie dostać 6 punktów. Można wybierać zarówno łatwiejsze zadania, jak i trudniejsze, za większą liczbę punktów. Można również rozwiązać zadania za większą liczbę punktów, np. za 10. Jeśli wtedy poprawnie będą rozwiązane zadania za 5 punktów, to zostanie przydzielone 5 punktów. Jeśli będą poprawnie rozwiązane zadania za więcej niż 6 punktów, np. 8 czy 10, to taka osoba otrzyma maksymalnie 6 punktów.

Zadanie 7.0.1 — 1 pkt

Napisz pętlę, która wypisze wszystkie dwucyfrowe liczby podzielne przez 7. Kolejne liczby powinny być wypisane w jednym wierszu i porozdzielane pojedynczymi spacjami.

Zadanie 7.0.2 — 1 pkt

Napisz pętlę, która wypisze 20 elementów ciągu geometrycznego, który zaczyna się od 65536 i ma iloraz -1/2. Kolejne liczby powinny być wypisane w jednym wierszu i porozdzielane pojedynczymi przecinkami.
Kilka pierwszych elementów tego ciągu: 65536, -32768, 16384, -8192, 4096 itd.

Zadanie 7.0.3 — 1 pkt

Wypisz na konsolę 20 pierwszych liczb ciągu Fibonacciego. Liczby powinny być wypisane po jednej w każdym wierszu.

Wskazówka: jeśli macie zajęcia przed wykładem, to na wykładzie poznacie ciąg Fibonacciego, czyli ciąg 0, 1, 1, 2, 3, 5, 8, ..., w którym każdy kolejny wyraz jest sumą dwóch poprzednich.

Zadanie 7.0.4 — 1 pkt

Napisz pętlę, która obliczy liczbę wszystkich możliwych kombinacji „6” w grze liczbowej „Lotto”. Dla przypomnienia losujemy tam zbiór 6 liczb ze zbioru 49 liczb.

Zadanie 7.0.5 — 1 pkt

Napisz instrukcję warunkową, która będzie rozpoznawała kierunek świata w zmiennej kierunek, tzn. jeśli będzie tam zapisane `S`, to wypisze `Południe`, jeśli `N` to `Północ`, jeśli `W` to `Zachód`, a jeśli `E` to `Wschód`. Jesli będzie tam coś innego, to ma wypisać na konsolę `Inny`.

Zadanie 7.0.6 — 1 pkt

Napisz pętlę, która wypisze liczby od 1 do 10, każdą w oddzielnej linijce. Jeśli jest ona podzielna przez 2, niech wypisze ją 2 razy rozdzielając liczby przecinkiem np. 4, 4. Jeśli jest ona podzielna przez 3, niech wypisze ją 3 razy, rozdzielane przecinkami. Jeśli jest ona podzielna naraz przez 2 i 3, to niech wypisze ją jeden raz.

Zadanie 7.0.7 — 2 pkt

Oblicz szansę trafienia „3” w grze liczbowej „Lotto”. Dla przypomnienia losujemy tam zbiór 6 liczb ze zbioru 49 liczb.

Zadanie 7.0.8 — 2 pkt

Napisz pętlę, która dla danej listy słów `lista` wypisze w kolejnych wierszach ich skróty w postaci `<pierwsza litera>-<ostatnia litera> (<dlugosc slowa>)`. Np. dla listy `lista = ['Interdyscyplinarne', 'Centrum', 'Modelowania']` powinna wypisać:

```
I-e (18)
C-m (7)
M-a (11)
```

Wskazówka: wynik funkcji `len()` mierzącej długość słowa jest liczbą. Do rozwiązania tego zadania może Ci się przydać konwersja tej liczby na słowo (aby dało się ją skleić z innymi słowami). Do tego służy polecenie `str()`:

```
 > len('dudy')
=> 4
 > str(len('dudy'))
=> '4'
```

Zadanie 7.0.9— 2 pkt

Wypisz na konsoli wszystkie liczby pierwsze, które są mniejsze niż 1000, każdą w oddzielnej linijce.

Zadanie 7.0.10 — 3 pkt

Używając dwóch pętli **for** (jedna wewnątrz drugiej), napisz program, który dla wypisze na konsoli trójkąt Pascala do wybranej liczby linijek n.
Dla n = 6 wynik będzie następujący:

```
1,
1, 1,
1, 2, 1
1, 3, 3, 1
1, 4, 6, 4, 1
1, 5, 10, 10, 5, 1
```

Zadanie 7.0.11 — 3 pkt

Rozwiązywałeś Zadanie 11 z części matematycznej? Teraz policzymy na komputerze zadanie analogiczne. Mamy do wysłania (w sumie) 24 identycznych listów i musimy je wysłać do ośmiu osób. Na ile sposobów można to zrobić? Napisz również pętlę, która obliczy na ile sposobów można to zrobić, gdy będziemy mieli przypadek ogólny, czyli n listów oraz k osób.