Zadania na ćwiczenia - Python

23 stycznia 2020

Spis treści

Zasady oceniania

W sumie można dostać 110 punktów, w tym:

- zadania domowe: do 40 pkt
- zadania z ćwiczeń: do 30 pkt
- egzamin/projekt końcowy: do 30 pkt
- nieobowiązkowe dodatkowe zadania: do 10 pkt

Ocena końcowa zależy od sumy punktów, ale żeby zaliczyć przedmiot, trzeba mieć nie mniej niż 50% punktów w każdej z trzech pierwszych kategorii. Punkty dodatkowe mogą podwyższyć pozytywną ocenę.

- Zadania domowe i ewentualne zadania z ćwiczeń należy przesłać mailem przed kolejnymi zajęciami.
- Oceniany jest również styl (czytelność kodu, sensowne nazwy zmiennych i funkcji, zamieszczanie stosownych komentarzy, czytelne komunikaty wypisywane przez program itd.).

Ćwiczenia 1

Komunikacja z użytkownikiem, zmienne i podstawowe typy, operatory

Zadanie 1

Napisz program wypisujący dwa dowolne komunikaty na ekran.

Zadanie 2

Napisz program proszący użytkownika o imię i rok urodzenia, a następnie obliczający i wypisujący jego wiek.

Przykładowe wykonanie:

```
Podaj swoje imię:
Jasiu
Podaj rok urodzenia:
1989
Jasiu, masz 30 lat.
```

Zadanie 3

Napisz program proszący użytkownika o podanie dwóch liczb `a` i `b` i wypisujący ich sumę, różnicę, iloczyn, iloraz, $\sqrt{a+b}$ oraz a^b i b^a.

Zadanie domowe - układy współrzędnych (2 pkt)

Napisz program, który poprosi użytkownika o podanie współrzędnych kartezjańskich x, y, z punktu w przestrzeni, a następnie przeliczy je na współrzędne w układzie sferycznym i cylindrycznym i wypisze wyniki. Zadbaj nie tylko o poprawność wyników, ale także o czytelność kodu i komunikację z użytkownikiem. Użyj funkcji matematycznych z biblioteki `math`.

Ćwiczenia 2

instrukcja warunkowa, pętle
Materiały z przykładami:
`https://repl.it/repls/CaringFlawlessScientist`
lub
`wget https://www.fuw.edu.pl/~msroczynska/instr_print_if_loops.py`

Zadanie 4

Napisz program, który oblicza pole i obwód koła o promieniu podanym przez użytkownika. Wartość stałej π weź z biblioteki `math`. Promień nie może być ujemny. W przypadku podania liczby ujemnej, program powinien wypisywać komunikat informujący o błędnej wartości i nic nie liczyć.

Zadanie 5 (2 pkt)

Napisz program, który prosi o podanie współczynników równania kwadratowego $ax^2 + bx + c = 0$ i wypisuje rozwiązania równania (lub informację o braku rozwiązań).

Zadanie 6

Napisz program, który wylosuje dowolną liczbę całkowitą od zera do 10 (do losowania liczb użyj np. funkcji `randint` z biblioteki `random` `https://docs.python.org/3/library/random.html`), a następnie prosi użytkownika o jej zgadnięcie tak długo, aż ten poda poprawną wartość. Gdy program działa, rozszerz go np. o podawanie informacji za którym razem udało się zgadnąć lub o wskazówki typu "Podana przez ciebie liczba jest większa/mniejsza od wylosowanej".

Zadanie domowe - liczby Catalana (3 pkt)

Liczby Catalana (`https://en.wikipedia.org/wiki/Catalan_number`) to ciąg, który można wyrazić wzorem rekurencyjnym:

$$\begin{aligned} C_0 &= 1 \\ C_{n+1} &= \frac{4n+2}{n+2} C_n. \end{aligned} \tag{0.1}$$

Napisz program, który wypisuje wszystkie liczby Catalana mniejsze od miliona i podaje, ile jest wśród nich liczb parzystych i nieparzystych. *Uwaga na kolejność instrukcji w pętli - może mieć znaczenie!*

Ćwiczenia 3

Zadanie 7 (1 pkt)

Napisz program, który wypisze pierwsze N liczb naturalnych w kolejności rosnącej i malejącej w osobnych kolumnach:

```
0 5
1 4
2 3
3 2
4 1
5 0
```

Zadanie 8

Napisz program, który obliczy sumę pierwszych N wyrazów rozwinięcia w szereg Maclaurina funkcji e^x:

$$e^x = \sum_{n=0}^{\infty} \frac{x^n}{n!}$$

dla x zadanego przez użytkownika. Porównaj wynik z wartością obliczoną przez funkcję `exp` z biblioteki `math`. Poeksperymentuj z różnymi wartościami N, np. 5, 10, 15.
(1 pkt z puli punktów dodatkowych)* Jak to zrobić bez obliczania silni w programie?

Zadanie domowe (3 pkt)

Liczby doskonałe to takie, które są sumą swoich dzielników właściwych (`https://en.wikipedia.org/wiki/Perfect_number`). Napisz program, który dla podanej liczby naturalnej sprawdza, czy jest ona liczbą doskonałą.

Zadanie 9

Napisz program, który prosi o podanie liczby naturalnej, a następnie wypisuje z ilu cyfr składa się wpisana wartość, a także informację o sumie tworzących ją cyfr. Dla uproszczenia załóż, że podana liczba jest poprawna (czyli rzeczywiście naturalna).

Zadanie 10

Napisz program, który:

- utworzy pustą listę i wypełni ją pięcioma liczbami losowymi z dowolnego zakresu, np. $[-1, 1]$ (do generowania liczb możesz użyć np. funkcji `uniform` z modułu `random`, której wywołanie `uniform(a, b)` zwraca liczbę z przedziału $[a, b]$);
- wypisze po kolei wszystkie elementy listy wraz z indeksami w następujący sposób:

```
lista[0]: 0.1
lista[1]: -0.92
itd.
```

- wypisze długość listy, czyli ilość jej elementów;
- znajdzie i wypisze największy element listy oraz jego indeks.

Ćwiczenia 4

Pomocne materiały:

- `https://www.fuw.edu.pl/~msroczynska/instr_functions.py`
- `https://brain.fuw.edu.pl/edu/index.php/PPy3/Funkcje`
- `https://www.python.org/dev/peps/pep-0257/`

Zadanie 11

Napisz funkcję, która przyjmuje wartość temperatury w Kelvinach i zwraca wartość wyrażoną w stopniach Celsjusza: $T_C = T_K - 272.15$. W przypadku podania wartości ujemnej funkcja zwraca `None`. Przetestuj jej działanie.

Zadanie 12

Napisz funkcję, która zwraca sumę pierwszych N wyrazów rozwinięcia exp w szereg Maclaurina. Wartości x i N niech będą argumentami funkcji. Możesz wykorzystać kod napisany w zadaniu 7. Zadaj wartość domyślną N wynoszącą 50. Przetestuj działanie funkcji porównując wyniki z obliczonymi przez `exp` z biblioteki `math`.

Zadanie domowe (4 pkt)

Serie widmowe wodoru dane są poniższym wzorem (`https://pl.wikipedia.org/wiki/Serie_widmowe_wodoru`)

$$\frac{1}{\lambda} = R_\infty \left(\frac{1}{n_1^2} - \frac{1}{n_2^2} \right),$$

gdzie:

- λ - długość fali fotonu odpowiadającego przejściu z powłoki n_2 na powłokę n_1
- n_1, n_2 - numery orbitali, między którymi następuje przejście, $n_1 < n_2$
- $R_\infty \approx 0.0109737 nm^{-1}$ - stała Rydberga.

Napisz program, który wypisze długości fali kilku serii widmowych wodoru np. w taki sposób:

```
Linie widmowe dla n1 = 1:
n2 = 2 : lambda = 122 nm
n2 = 3 : lambda = 103 nm
n2 = 4 : lambda = 97 nm
Linie widmowe dla n1 = 2:
n2 = 3 : lambda = 656 nm
n2 = 4 : lambda = 486 nm
n2 = 5 : lambda = 434 nm
Linie widmowe dla n1 = 3:
n2 = 4 : lambda = 1875 nm
n2 = 5 : lambda = 1281 nm
n2 = 6 : lambda = 1094 nm
```

W swoim programie zawrzyj funkcję, która oblicza λ dla zadanego n_1 i n_2. Jeśli spełniony jest warunek $n_1 < n_2$, to funkcja powinna zwracać długość fali obliczonej z podanego wyżej wzoru, a w przeciwnym razie wypisywać komunikat o niepoprawnych argumentach i zwracać 0 lub `None`. Pętlę wypisującą obliczone wartości umieść w osobnej funkcji, której argumentami są maksymalna wartość n_1 oraz to, ile długości fali wypisać dla każdej serii. W razie podania niepoprawnych wartości, funkcja powinna wypisać odpowiedni komunikat i nic nie liczyć (może zawierać pustą instrukcję return). Pod definicjami obu funkcji wpisz wywołanie drugiej z nich dla dowolnych sensownych wartości i sprawdź, czy działa poprawnie. Wartości argumentów, z którymi ma być wywołana funkcja wypisująca mogą być podawane z klawiatury przez użytkownika lub wpisane w kodzie.
Uwagi:

- Słowo `lambda` jest jednym z słów kluczowych Pythona (podobnie jak np. `for` czy `if`) i nie można nazwać tak zmiennej!
- Treść polecenia jest zdecydowanie dłuższa od kodu, który trzeba napisać :)
- Użycie nieco mniej/bardziej dokładnej wartości stałej Rydberga może troszkę zmienić wyniki.
- Długości fali można wypisywać jako liczby całkowite używając np. funkcji `format`, można też użyć przybliżenia za pomocą wbudowanej funkcji `round`.
- Chętni mogą nieco rozszerzyć zadanie i dostać punkty z puli dodatkowych, np. o obliczanie serii widmowych atomów wodoropodobnych, wprowadzenie minimalnej wartości n_1 itp.

Zadanie 13

Napisz dwie funkcje obliczające silnię liczby podanej jako argument: w sposób iteracyjny i rekurencyjny. Przetestuj ich działanie. Możesz porównać wynik z wynikiem obliczonym przez funkcję `factorial` z biblioteki `math`.

Ćwiczenia 5

Zadanie 14 (4 pkt)

Napisz dwie funkcje funkcje zwracające n-ty wyraz ciągu Fibonacciego. Pierwsza powinna obliczać go iteracyjnie (2 pkt), a druga rekurencyjnie (2 pkt). Następnie korzystając z modułu `timeit` porównaj czas działania obu funkcji w zależności od wartości n (n od zera do ok. 30 - 40) i przedstaw je na wykresie.

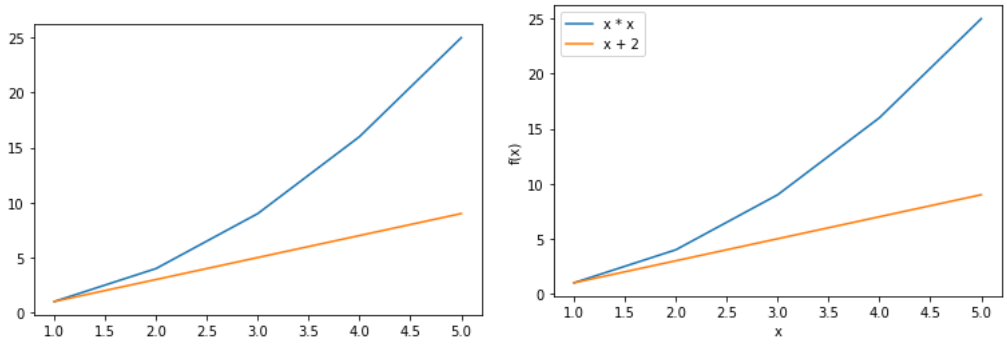
Czas działania funkcji. Najprościej można go zmierzyć w poniższy sposób:

```
import timeit
def f():
    # instrukcje
start_time = timeit.default_timer()
f()
end_time = timeit.default_timer()
print("Wykonanie␣f()␣zabrało␣{}s.".format(end_time - start_time))
```

Ten czas może być za każdym razem inny. Często lepiej mierzyć uśredniony czas dzięki wywoływaniu funkcji wiele razy, ale tutaj nie jest nam to potrzebne. Poza tym, w tym przypadku powtórzenie wywołania kilkaset razy mogłoby zabrać sporo czasu...

Rysowanie wykresu. Do narysowania wykresu wykorzystamy bibliotekę `matplotlib`, którą nieco dokładniej jeszcze omówimy trochę później. W tym zadaniu chcemy narysować wykres, w którym na osi poziomej będzie wartość n, a na osi pionowej czas, który zajęło wywołanie funkcji dla danego n. Dane powinny być przygotowane w postaci dwóch list:

- pierwsza lista to wszystkie wartości n,
- druga lista to czasy wykonania funkcji dla każdgo n.



Rysunek 1: Wykresy wykonane za pomocą kodów z zadania 14.

Listy te muszą mieć tą samą długość. Na jednym wykresie można umieścić więcej niż jedną krzywą. Poniżej dla kilku liczb całkowitych rysujemy wykres $y = x^2$ oraz $y2 = x + 2$.

```
import matplotlib.pyplot as plt
x = [1, 2, 3, 4, 5]
y = [1, 4, 9, 16, 25]
y2 = [1, 3, 5, 7, 9]
plt.plot(x, y)
plt.plot(x, y2)
```

Można dodać nazwy osi i legendę:

```
import matplotlib.pyplot as plt
x = [1, 2, 3, 4, 5]
y = [1, 4, 9, 16, 25]
y2 = [1, 3, 5, 7, 9]
plt.plot(x, y, label = "x␣*␣x")
plt.plot(x, y2, label = "x␣+␣2")
plt.xlabel("x")
plt.ylabel("f(x)")
plt.legend()
```

Zadanie domowe (3 pkt)

Napisz funkcję, która dla zadanego argumentu (którym może być lista, krotka lub inna sekwencja) zwraca indeks i wartość najmniejszego elementu oraz indeks i wartość największego elementu. Następnie, używając składni *list comprehension* utwórz kilkunastoelementową listę wypełnioną losowymi liczbami całkowitymi i przetestuj na niej swoją funkcję.
Uwaga. Python posiada wbudowane funkcje `min, max` itp. do znalezienia maksimum, minimum oraz indeksów. Można (ale nie jest to obowiązkowe) ich użyć żeby sprawdzić, czy napisana przez nas funkcja daje poprawne wyniki. Jednak w funkcji nie należy ich używać - należy przejść wszystkie elementy listy pętlą `for`, porównywać z czym trzeba i w ten sposób znaleźć szukane wartości.

Ćwiczenia 6

Krotki, słowniki, zbiory
Materiały:

- 20-minutowy filmik podsumowujący własności list, zbiorów itd.: `https://www.youtube.com/watch?v=R-HLU9Fl5ug` (warto poszukać też filmików wyjaśniających inne rzeczy z Pythona - na YouTube jest ich mnóstwo)
- Rozdział 6 dot. typów dynamicznych z książki Marka Lutza pt. "Python. Wprowadzenie.". Książka jest dostępna w bibliotece.

Zadanie 15

Napisz funkcję, która dla dowolnej sekwencji podanej jako argument, obliczy ile razy wystąpił w niej każdy z elementów i zwróci te dane jako wynik. Następnie wypisz liczbę wystąpień każdego elementu na ekran zaczynając od elementu najmniejszego. Niezbędne sortowanie powinno się odbyć już poza funkcją.
Wersja podstawowa: w funkcji utwórz słownik, w którym kluczami są poszczególne elementy sekwencji, a wartościami - liczba ich wystąpień.
Dodatkowo dla chętnych: Możesz stworzyć drugą funkcję, która użyje `Countera` (`https://docs.python.org/3/library/collections.html#collections.Counter`). `Counter` obsługuje składnię słownika, dlatego samo wypisywanie na ekran powinno działać dokładnie tak samo jak dla funkcji z wersji podstawowej.

Do sortowania można użyć wbudowanej funkcji `sorted`.

Np. dla napisu `"abzkfhiohfnlaanfnffnkkdamd"` na ekran powinno się wypisać:

```
a : 4
b : 1
d : 2
f : 5
h : 2
i : 1
k : 3
l : 1
m : 1
n : 4
o : 1
z : 1
```

Zadanie 16

Utwórz dwa zbiory i wypełnij je kilkoma losowymi liczbami całkowitymi z dowolnego przedziału, np. od 1 do 10 (przedział nie powinien być zbyt szeroki, żeby zbiory mogły mieć jakąś część wspólną). Następnie wypisz ich: sumę, różnicę i część wspólną, a także te elementy, które pojawiają się tylko w jednym z nich.

- Jeśli przy każdym uruchomieniu programu chcemy dostawać te same wartości z generatora liczb losowych, to trzeba na początku programu ustawić stałą wartość tzw. "ziarna" (*seed*). Można to zrobić pisząc `random.seed(0)`.
- Liczby losowe można wygenerować za pomocą `random.randint(a, b)`, gdzie `a` i `b` to górne i dolne ograniczenie przedziału.

Zadanie domowe

- Zapoznaj się z materiałami do tych zajęć. Istotne jest, żeby zrozumieć różnicę między obiektami *mutable* i *immutable*. Nie trzeba znać na pamięć wszystkich metod listy, zbioru itd., wystarczy się orientować, co można z nimi zrobić :)
- Na jednych z najbliższych zajęć będziemy korzystać z `jupyter-notebook` w wersji online, czyli `Google Colaboratory` (`https://colab.research.google.com`). Potrzebne do tego jest konto Google lub zainstalowany `jupyter notebook` na własnym komputerze. Upewnij się, że masz konto Google, jeśli nie - załóż je. Jeśli chcesz pracować offline na własnym komputerze, możesz zainstalować `jupyter-notebook`. Najwygodniej to zrobić instalując pakiet Anaconda (`https://www.anaconda.com/distribution/`). Poza `jupyter-notebook` przyda się również `NumPy`, `SciPy` i `Matplotlib`, które także są zawarte w pakiecie Anaconda.

Zadanie 17

Czy następujące programy są poprawne? Jeśli tak, to co wypiszą instrukcje `print` i dlaczego? Jeśli nie, to gdzie są błędy?

1. ```
   t = (1, 2, 3, 4)
   t[0] = 8
   print(t)
   ```

2. ```
   t = (1, 2, 3, 4)
   t.append(5)
   print(t)
   ```

3. ```
   t = "hehe"
   w = t*5
   print(w)
   ```

4. ```
   t = {}
   print(type(t))
   ```

5. ```
   t = {}
   t[1] = 10
   t[199] = 'c'
   print(t)
   ```

6. ```
   a = [1, 2, 3]
   b = a
   b[2] = 10
   print(a)
   print(b)
   ```

7. ```
   L = [1,2,3]
   for element in L:
       element = 5
   print(L)
   ```

8. ```
   L = [1,2,3]
   for i in range(len(L)):
       L[i] = 5
   print(L)
   ```

9. ```
   if True:
       x = 10
   print(x)
   ```

10. ```
    for i in range(5):
        pass
    print(i)
    ```

11. ```
    def f():
        print(a)
    a = 8
    f()
    ```
    ```

12.
```
def f():
    a = 15
a = 8
f()
print(a)
```

13.
```
def f():
    global a
    a = 15
a = 8
f()
print(a)
```

14.
```
def f():
    global a
    a = 15
a = 8
print(a)
```

15.
```
def f(L1):
    L1[0] = 15
L = [1,2,3]
f(L)
print(L, L1)
```

16.
```
def f(L1):
    L = L1
L = [1,2,3]
f([4,5,6])
print(L)
```

17.
```
def f(L1):
    global L
    L = L1
L = [1,2,3]
f([4,5,6])
print(L)
```

18.
```
def f(L1):
    L = L1[:]
    L[0] = 15
L = [1,2,3]
f([4,5,6])
print(L)
```

19.
```
def f(L1):
    L = L1[:]
    L[0] = 15
L = (1,2,3)
f((4,5,6))
print(L)
```

Ćwiczenia 7

Materiały:

- Dla zainteresowanych: `map`, `filter`, `reduce` i użycie wyrażeń lambda: `https://www.youtube.com/watch?v=cKlnR-CB3tk`

Zadanie 18

Napisz program, który dla zadanej listy par wypisze tą listę posortowaną ją według drugiego elementu, czyli np. dla listy zawierającej imiona i nazwiska kilku osób:

```
ludzie = [('Jan', 'Kowalski'),
    ('Grzegorz', 'Brzęczyszczykiewicz'),
    ('Jacek', 'Placek')]
```

wypisze posortowaną według nazwisk.
Wykorzystaj wbudowaną funkcję `sorted` i jej argument opcjonalny `key`, którym powinna być funkcja zwracająca drugi element pary. Wypróbuj dwie wersje: wykorzystując jako `key` funkcję zdefiniowaną za pomocą `def` oraz za pomocą wyrażenia `lambda`.

Program powinien wyglądać mniej więcej tak (trzeba uzupełnić tam, gdzie wielokropki):

```
ludzie = [('Jan', 'Kowalski'),
    ('Grzegorz', 'Brzęczyszczykiewicz'),
    ('Jacek', 'Placek')]
print(ludzie)

# wersja z osobną funkcją zadaną przez def
def ... # funkcja zwracająca drugi element sekwencji (ten o indeksie 1!)

ludzie_sorted = sorted(ludzie, key = ...) # używamy tej funkcji jako key
print(ludzie_sorted)

# wersja z lambdą
ludzie_sorted = sorted(ludzie, key = ...) # definiujemy odpowiednią lambdę
print(ludzie_sorted)
```

Zadanie 19

Napisz funkcję, która mierzy czas wykonania innej funkcji za pomocą `timeit` tak, jak w przykładzie. Argumentami tej funkcji jest nazwa funkcji, którą chcemy wykonać oraz wartości jej argumentów. Chodzi o to, by można było wykorzystać ją w następujący sposób:

```
def czas_wykonania#...

def f1(x):
    #...
def f2(x1, x2, x3):
    #...
t1 = czas_wykonania(f1, 2) # mierzy czas wykonania f1(2) i przypisuje go do t1
t2 = czas_wykonania(f2, 3, 4, 5) # mierzy czas wykonania f2(3,4,5)
                                 # i przypisuje go do t2
```

Zadanie domowe (3 pkt)

Wykorzystaj funkcję napisaną w poprzednim zadaniu w kodzie z zadania 14 (ciąg Fibonacciego).
**Dla chętnych (1 pkt z puli dodatkowych):* Możesz poszukać w internecie innych implementacji funkcji zwracających n-ty wyraz ciągu Fibonacciego i porównać je z tymi, które już masz.

Ćwiczenia 8

Materiały: `https://brain.fuw.edu.pl/edu/index.php/PPy3/Wej%C5%9BcieWyj%C5%9Bcie`

Zadanie 20 (3 pkt)

Napisz program, który w pętli prosi o podanie liczby naturalnej i zapisuje każdą podaną liczbę do odpowiedniego pliku: parzyste do `even.txt`, a nieparzyste do `odd.txt`.
Program kończy się, gdy wpisane zostanie 0.
Pliki powinny być zapisywane w nowym folderze, który zostanie utworzony przy uruchomieniu programu (o ile jeszcze nie istnieje). Aby to zrobić, trzeba użyć modułu `os`. Przydatne funkcje:

- `os.path.exists(mypath)` - sprawdza, czy folder istnieje
- `os.mkdir(mypath)` - tworzy folder w zadanej ścieżce `mypath`
- `os.getcwd()` - aktualny katalog

Uruchomienie programu po raz kolejny powinno powodować, że nowe liczby zostaną dopisane do istniejących plików lub zostaną utworzone nowe pliki.

Zadanie 21

Napisz program, który wczyta pliki utworzone w poprzednim zadaniu i obliczy sumę oraz średnią arytmetyczną liczb znajdujących się w każdym z nich.

Zadanie domowe (3 pkt)

Plik z liczbami do zadania: `https://www.fuw.edu.pl/~msroczynska/templates/99_numbers.txt`
Była sobie lista stu nieposortowanych liczb naturalnych od 1 do 100, na której żadna z liczb się nie powtarzała. Tej stuelementowej listy jednak nie znamy. Wiemy tylko, że usunięto z niej jedną z wartości i zostało 99 różnych liczb. Ta skrócona już o jeden element lista jest zawarta w pliku dostępnym pod linkiem powyżej. Napisz program, który wczyta zawartość tego pliku i sprawdzi jakiej liczby brakuje, a następnie wypisze stosowną informację. Można używać dowolnych funkcji i struktur danych dostępnych w Pythonie, ale nie jest tutaj potrzebne nic ponad to, co już się na zajęciach pojawiło :)

Ćwiczenia 9

Na tych zajęciach pracujemy w notebooku Google Colaboratory. Jest on dostępny pod linkiem: `https://colab.research.google.com/drive/16XTUS-IF5qm6P7zb80mDUoSbCtY1iYri`
Aby zacząć z nim pracować:

1. otwórz adres podany w linku
2. zaloguj się na swoje konto Google
3. zapisz kopię na swoim Dysku Google (*File* → *Save a copy in Drive*)
4. możesz dowolnie edytować swoją kopię

Ćwiczenia 10

Na tych zajęciach również pracujemy w notebooku Google Colaboratory. Jest on dostępny pod linkiem: `https://colab.research.google.com/drive/1mpklJbqkU1dPyMu7Xjeqmv1RI8fIIcNi`

Ćwiczenia 11

Materiały:

- `https://python.swaroopch.com/oop.html` - trochę na temat klas

Zadanie 22

Szkielet rozwiązania: `https://repl.it/repls/BlindEuphoricOperation`
Napisz i przetestuj klasę reprezentującą kota (lub jakiekolwiek zwierzę :)). Rozwiązanie zapisz w pliku `kot.py`. Każdy kot ma:

- imię,
- swoje ulubione zabawki,
- zabawki, których nie znosi,

a ponadto potrafi:

- przywitać się i przedstawić,
- powiedzieć, jakie są jego ulubione/nielubiane zabawki i ile ich jest,
- polubić napotkaną zabawkę,
- obrazić się na jakąś zabawkę, którą dotąd lubił.

Kot może polubić zupełnie nową zabawkę, ale może też nagle się na którąś "odobrazić" i zabawka z nielubianej stanie się lubianą. Może też być odwrotnie: nagle przestanie mu się podobać jedna z tych, które dotąd lubił. Do nowo napotkanych zabawek podchodzi z ufnością, czyli może je polubić, ale nie ma tak, że zobaczy i od razu nie znosi.
Uwaga: Na liście lubianych i nielubianych zabawek żaden element się nie powtarza! Koty są też na tyle ogarnięte, że nie mogą danej zabawki jednocześnie lubić i nie znosić.

Zadanie 23

Szkielet rozwiązania: `https://repl.it/repls/SoftRequiredIrc`
Przykładowe dane: `https://repl.it/repls/SpectacularEnragedKnowledge`
Korzystając z klasy Kot, stwórz małą symulację, w której jest kot idący sobie gdzieś wzdłuż korytarza. Po drodze mija dużo pudełek z losowymi zabawkami (ale w każdym pudełku jest tylko jeden rodzaj zabawek). Kiedy napotyka pudełko, widzi trzymane tam zabawki i losowo wybierana jest jedna z poniższych opcji:

- kot przechodzi obok zabawki obojętnie,
- jest zachwycony zabawką i uznaje, że ją lubi,
- stwierdza, że jeśli ją lubił, to najwyższy czas ją teraz znienawidzić.

Później kot idzie dalej, znów sprawdza co tam jest i decyduje co robić. Symulacja kończy się po zadanej z góry liczbie iteracji (np. 100). Po zakończeniu kot mówi nam, które zabawki polubił, a które nie i ile ich było. Dla uproszczenia, zakładamy że kot napotyka pudełko na każdym kroku. Bywa, że w wielu pudełkach są te same zabawki.
Lista zabawek jest wczytywana z pliku, który najpierw trzeba stworzyć na swoim komputerze i wypełnić jakąś zawartością. Można zrobić swoją listę zabawek albo wykorzystać tą podaną pod linkiem podanym w materiałach. Funkcja wczytująca plik i zwracająca listę znajdujących się tam elementów jest już gotowa w szkielecie rozwiązania.

Zadanie domowe dla chętnych

Zmodyfikuj/rozszerz symulację według własnego pomysłu :)
Zadanie nie jest obowiązkowe, można dostać kilka punktów z puli dodatkowych.

Ćwiczenia 12

Materiały:

- `https://www.kodolamacz.pl/blog/wyzwanie-python-5-zaawansowane-aspekty-programowania-obiekto`

Zadanie 24

Szkielet programu:`https://repl.it/@mksroczynska/Forest`
Uzupełnij szkielet programu o brakujące atrybuty i treści metod klas `Bobr`, `Drzewo` i `Las`.
W nowo posadzonym lesie równiutko rosną drzewa, ale nagle zaczyna grasować w nim bóbr. W tym momencie zaczyna się ewolucja lasu, którą należy zasymulować. Bóbr codziennie wyrusza na łowy i podgryza losowe drzewo. Wszystkie pozostałe drzewa codziennie trochę rosną - ich średnica zwiększa się o 1%. Podgryzione drzewo zaczyna rosnąć dopiero kolejnego dnia. Bóbr jednak trochę się męczy przy gryzieniu drzew. Im grubsze drzewo, tym więcej wysiłku kosztuje go jego nadgryzienie. Bóbr dzielnie walczy i cała historia kończy się gdy uda mu się powalić pierwsze drzewo (czyli średnica drzewa zmniejszy się do zera) lub gdy bóbr się zmęczy i jego siły zmaleją do zera.
Każde drzewo ma na początku średnicę równą $1m$. Siła bobra reprezentowana jest przez liczbę i na początku wynosi 2. Podgryzienie drzewa sprawia, że zapas sił bobra zmniejsza się o wartość równą 7% długości średnicy w metrach. Podgryzienie drzewa zmniejsza jego średnicę aż o $0.2m$.

Zadanie 25

Szkielet programu: `https://repl.it/@mksroczynska/DomoweZoo`
Każdy zwierzak przygarniany do domu ma swoje imię i wiek. Poza tym, każdy z nich umie się przedstawić. Napisz klasę `Zwierze` zawierającą odpowiednie pola i metody.
Każdy zwierzak, gdy się przedstawia, to oprócz swojego imienia i wieku, wypisuje także informację o tym, jakiego gatunku zwierzakiem jest.
Napisz dwie klasy dziedziczące z klasy `Zwierze` reprezentujące zwierzaki domowe, np. psa czy chomika. Rozszerz metodę, która wypisuje informacje o zwierzątku w odpowiedni sposób.
Następnie stwórz kilka zwierzaków i utwórz obiekt klasy `DomoweZoo`. Dodaj do niego zwierzaki za pomocą metody `dodaj`. Następnie wywołaj metodę `wypiszZwierzaki`, która sprawia, że każdy domowy pupil musi się przedstawić.

Ćwiczenia 13

Zadanie 26

Napisz program, który uruchomiony z terminala poleceniem `python3 program.py plik1.txt plik2.txt plik3.txt` wypisze zawartość plików `plik1.txt, plik2.txt, plik3.txt`, a jeśli któregoś z nich nie da się otworzyć, wypisze odpowiednią informację.

Zadanie 27

Napisz program, który dzieli dwie liczby i wypisuje ile czasu to zajęło. Program powinien wypisywać czas zawsze, również jeśli wystąpił błąd dzielenia przez zero lub inny. Jeśli żaden błąd nie wystąpił, program powinien wypisać również wynik dzielenia.

Zadanie 28 (4 pkt)

Napisz funkcję, która zamienia miejscami dwa elementy listy. Funkcja przyjmuje jako argument listę oraz indeksy elementów, które chcemy ze sobą zamienić. Napisz również kod, który tą funkcję wywołuje i obsługuje kilka potencjalnych typów wyjątków, jakie mogą być rzucane przez tą funkcję w przypadku wywołania jej z nieprawidłowymi argumentami. Ponadto, jeśli lista przekazana jako argument okaże się pusta, funkcja powinna rzucać wyjątek `ValueError`.
(* *2 pkt z puli dodatkowych*) Napisz własną klasę dla wyjątku rzucanego gdy lista jest pusta, np. `EmptyListError` albo `EmptyContainerError`.

Ćwiczenia 14

Na tych zajęciach będziemy ponownie korzystać z Google Colaboratory.
Link do notebooka: `https://drive.google.com/open?id=10_uJP6OAheCAz2AjTJGDQIlEGJoYWTqq`
Link do pliku z danymi: `https://drive.google.com/open?id=1l9IDLgeBy2E3r2cLGBm6rYO_ExNRSr_1`

Uwaga: Plik z danymi będziemy wczytywać w notebooku. Należy go zapisać na swoim dysku Google, najlepiej w tym samym folderze co notebook (czyli klikamy w link, logujemy się na swoje konto Google i klikamy "Dodaj do mojego dysku").

Projekt końcowy

Sprawy organizacyjne

- Program należy przesłać mailem przed rozmową zaliczeniową. Najlepiej najpóźniej dzień przed (czyli 2 lutego 2020). Rozmowa potrwa ok. 20 minut.
- Pod poniższym linkiem

  `https://docs.google.com/document/d/1nL886pUxsX7Y3auDsHGXryQWBn5Z4C_ZAv5gdG9PJhY/edit?usp=sharing`

  można zapisać się na konkretną godzinę. Pod tabelką z godzinami można również wpisywać wszelkie pytania odnośnie projektu, na które odpowiem także w pliku dostępnym pod linkiem. Można też pytać albo umówić się na konsultacje przez email.
- Projekt można robić samodzielnie lub w parach. Liczba osób w grupie nie wpływa na ocenę. Pisząc program w parach trzeba mieć pojęcie również o tym co napisał(a) kolega/koleżanka.
- Projekt robimy w domu.
- Poprawa oceny jest możliwa w sesji poprawkowej, w razie potrzeby - proszę o kontakt mailowy.

Treść zadania

Treść zadania jest o wiele dłuższa niż kod, który trzeba napisać (cały program zajmuje ok. 150-200 linijek - choć oczywiście to bardzo zależy) :)
W dużym kwadratowym ogrodzie żyje gromadka kotów i sporo myszy. Od rana koty uganiają się za myszami i wydeptują ścieżki. Czasem kot i mysz mogą spotkać się w jednym miejscu. Jak kończy się taka sytuacja? To zależy (opis możliwych scenariuszy poniżej). Wieczorem na trawie widać ścieżki wydeptane przez zwierzęta. Na początku symulacji umieszczamy koty i myszy w ogródku. Lista położeń początkowych (czyli położeń kocich pudełek/mysich norek) wszystkich kotów i myszy wczytywana jest z plików, których opis znajduje się nieco dalej. Na jej podstawie należy stworzyć wszystkie obiekty reprezentujące zwierzęta.

W ciągu dnia zarówno koty, jak i myszy poruszają się po ogrodzie, każdy na swój sposób:

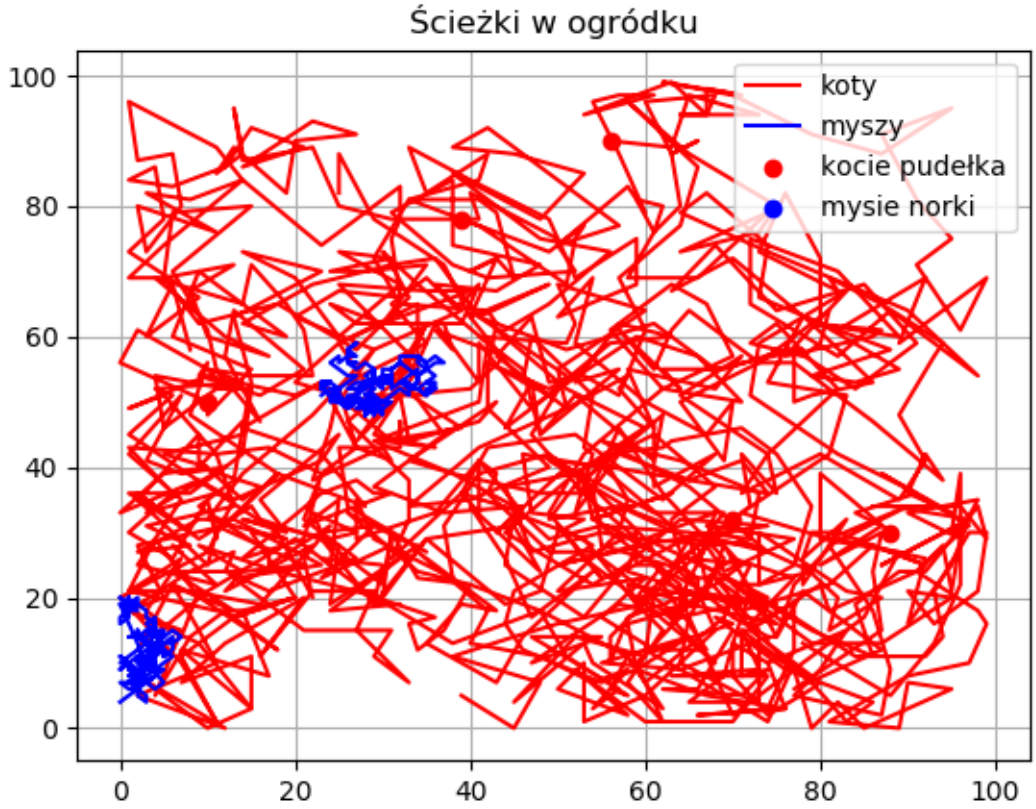
- *Myszy* są małe, więc robią małe kroczki. W jednym momencie mogą co najwyżej zmienić swoją pozycję (x, y) o 1 (lub nie zmienić wcale) w każdym kierunku. Wyjątek stanowi ucieczka, wtedy mysz nabiera supermocy i teleportuje się do swojego schronienia, niezależnie jak to daleko.
- *Koty Przeciętniaki* oraz *Koty Leniuchy* poruszają się w losowym kierunku zmieniając położenie o co najwyżej 10, zarówno w kierunku x, jak i y.
- *Kociaki* poruszają się w losowym kierunku, zmieniają swoją pozycję (każdą współrzędną) o co najwyżej 5, ale nigdy nie oddalają się od swojego domu o odległość większą niż 100. Jeśli nowe położenie sprawiłoby, że kociak oddali się od domu o ponad 100, to zamiast tego wraca do poprzedniego położenia.

Uwaga: Każdy zwierzak pamięta wszystkie swoje położenia.
Jeśli mysz znajdzie się w odległości mniejszej niż 4 od kota, to następuje ich spotkanie. Spotkanie kota i myszy może przebiegać różnie:

- *Koty Przeciętniaki*, których jest najwięcej, po prostu zawsze trącają mysz łapką, ona teleportuje się do swojego domku, a kot jest zadowolony.
- *Kociaki* są bardzo młode i praktycznie zawsze boją się myszy, a gdy ją spotkają, natychmiast teleportują się do swojego pudła. Mysz nadal normalnie spaceruje sobie po ogródku. Jednak jeżeli jakimś cudem spotkają mysz w pobliżu swojego pudełka (w promieniu 50), wtedy stają się odważne i potrafią przestraszyć gryzonia na tyle, że tym razem to on teleportuje się natychmiast do swojej kryjówki.

- *Koty Leniuchy* są względnie niegroźne. Gdy spotkają mysz, nawet w swoim mieszkanku, często zupełnie się nią nie przejmują i po spotkaniu każde z nich idzie w swoją stronę jak gdyby nic się nie stało. Czasem jednak kot trąca mysz łapką, ta ucieka (teleportuje się) do kryjówki, a kot jest zadowolony, że pogonił gryzonia. To, czy kot jest akurat zainteresowany myszą, jest losowe. Zależy jednak od liczby przegonionych już myszy. Im jest ich więcej, tym kotu jednak bardziej się chce. Prawdopodobieństwo zainteresowania wynosi $\frac{1}{1+e^{-0.1n}}$, gdzie n to liczba dotychczas przegonionych myszy.

Cały dzień trwa kilkaset iteracji, a cała symulacja to jeden dzień. Po zakończeniu symulacji chcemy zobaczyć wszystkie ścieżki wydeptane w trawie przez koty i myszy.
Pliki z danymi. Za pomocą dowolnego programu (lub ręcznie) należy stworzyć pliki tekstowy, który w każdej linijce zawiera dwie liczby oddzielone spacją - położenia początkowe x i y jednego zwierzaka. Dane dla każdego typu zwierząt powinny znaleźć się w osobnym pliku (3 typy kotów + myszy $\rightarrow$ w sumie 4 pliki). Położenia powinny być liczbami całkowitymi nieujemnymi, żaden zwierzak nie powinien też znajdować poza ogrodem. Należy stworzyć po kilka zwierząt każdego typu.
Po zakończeniu symulacji chcemy zobaczyć ścieżki wydeptane przez wszystkie zwierzęta. Jako że każde zwierzę pamięta wszystkie swoje położenia, wystarczy narysować wykres przedstawiający kolejno połączone punkty. Można to zrobić używając Matplotlib i polecenia `plot`.
Przykładowy wykres:



Zaliczenie projektu

Aby zaliczyć projekt (na 50%) należy co najmniej:

- zaimplementować odpowiednie klasy, które umożliwią tworzenie myszy i jednego typu kotów wraz z metodami, które pozwalają na ich poruszanie się;
- utworzyć obiekty na podstawie danych o położeniach początkowych wczytanych z pliku;
- napisać pętlę, w której koty i myszy zmieniają swoje położenia oraz następuje sprawdzanie czy nie ma jakiegoś spotkania kota z myszą, a jeśli jest - wykonuje się kod przewidziany dla takiej sytuacji zgodnie z opisem.

Oczywiście należy też wiedzieć co w tym programie się dzieje, po co napisaliśmy jakąś klasę czy funkcję itp. :)
W projekcie nie jest wymagane łapanie wyjątków. Można również założyć, że pliki z danymi są poprawne. Rozmiar ogrodu i liczba iteracji powinny mieć wartości rzędu 100, ale można zrobić dowolne (byle sensowne :)). Można je wczytać z osobnego pliku, ale w zupełności wystarczy wpisanie ich w kodzie.