Tematy na dzisiejsze zajęcia

- Zmienna liczba argumentów funkcji - args i kwargs
- Importowanie modułów wbudowanych
- Dokumentacja Pythona
- [tworzenie własnych modułów i pakietów]
- Praca z plikami - odczyt i zapis
- Obsługa wyjątków

Zmienna liczba argumentów funkcji - *args

```
def fun(*args):
    for item in args:
        print(item)
```

`fun(1,2)` -> 1, 2

`fun(1,2,10,20,30)` -> 1, 2, 10, 20, 30

“rozwinięcie” listy jako kolejnych argumentów:

```
tab = [1,2,4,8]
```

`fun(*tab)` -> 1, 2, 4, 8

Zmienna liczba argumentów funkcji - **kwargs

```
def fun(**kwargs):
    for item in kwargs:
     print(item, kwargs[item])
```

lub:

```
def fun(**kwargs):
    for key,val in kwargs.items():
     print(key, val)
```

`fun(aaa=100, bbb=1000, ccc=9999)` -> aaa 100, bbb 1000, ccc 9999

`fun(100, 1000, 9999)` -> BŁĄD! (muszą mieć nazwy)

"rozpakowanie" słownika jako kolejnych argumentów:

```
items = {"a": 10, "b": 20}
fun(**items)
```

-> a 10, b 20

Importowanie modułów

Aby w Pythonie użyć “czegoś dodatkowego” - modułu, należy to zaimportować. Na początku programu (zalecane, a obowiązkowo przed pierwszym użyciem) wpisujemy linijkę:

```
import nazwa_modułu
```

np.

```
import random
import math
import os
...
```

Moduł random

```
import random
```

Moduł random pozwala na używanie generatorów liczb pseudolosowych, dzięki czemu możemy m.in. losować jakąś liczbę lub element tablicy.

Proste i przydatne funkcje z modułu random:

`random.randint(A, B)` - zwraca losową liczbę całkowitą z przedziału [A, B]
`random.random()` - zwraca losową liczbę z przedziału [0.0, 1.0)
`random.choice(ARRAY)` - zwraca losowy element z listy ARRAY
`random.sample(k, population)` - zwraca k elementów bez powtórzeń z populacji (“duży lotek”)

...oraz wiele funkcji do losowań z rozkładów statystycznych

Moduł math

`import math`

Do bardziej zaawansowanych operacji matematycznych służy moduł math.

Najpotrzebniejsze funkcje:

`math.sqrt(N)` - √N (pierwiastek z liczby N)

`math.pow(N,a)` - N^a (N do potęgi a)

`math.floor(N)` - zaokrąglenie **N** w dół

`math.ceil(N)` - zaokrąglenie **N** w górę

`math.trunc(N)` - usunięcie części ułamkowej z **N**

`math.sin(x), math.cos(x), math.tan(x), itd.` - funkcje trygonometryczne z **x**

`math.degrees(x), math.radians(x)` - przeliczenie **x** radianów na stopnie / **x** stopni na radiany

`math.log(x, a)` - logarytm o podstawie **a** / logarytm dziesiętn z liczby **x**

`math.exp(x)` - wartość **e^x**

`math.pi, math.e -` stałe, wartości liczby pi oraz e (uwaga, nie funkcje!)

Moduły os i shutil

`import os`
Moduł os to operacje powłoki systemowej, np. zmiana folderu (cd), aktualna ścieżka (pwd), zawartość katalogu (ls/dir), także operacje na plikach

`import shutil`
Moduł shutil służy do prostych operacji na plikach i katalogach: kopiowanie, przenoszenie, zmiana nazwy, usuwanie

np.
`os.chdir(dst)`- zmiana folderu (cd)
`os.mkdir(name)`-utworzenie nowego folderu (mkdir)
`os.getcwd()` - aktualna ścieżka (pwd)
`os.listdir()` - lista plików w obecnej lokalizacji (ls)
`os.getlogin()` - aktualny zalogowany użytkownik (whoami)
`shutil.copy(src, dst)` - kopiowanie plików
`shutil.copytree(src,dst)` - kopiowanie folderów
`shutil.move(src, dst)` - przenoszenie plików lub folderów
`shutil.rmtree(dir)` - usuwanie katalogu (działa tylko jeśli ma odpowiednie prawa!)

Skąd to wszystko wiedzieć? :)

https://docs.python.org/3/ - dokumentacja Pythona
https://docs.python.org/3/library/ - biblioteka standardowa Pythona
(czyli wszystko co można zaimportować bez dodatkowych instalacji)
np.
https://docs.python.org/3/library/math.html
https://docs.python.org/3/library/random.html
https://docs.python.org/3/library/os.html
https://docs.python.org/3/library/shutil.html

Specyfika dokumentacji Pythona: jest obszerna, czytelna i zawiera sporo przykładów!

Moduł time

Do operacji związanych z czasem i zegarem służy moduł time

```
import time
```

Jednym z najczęstszych zastosowań jest funkcja `time.sleep(n)`,
która “zamraża” wywołanie programu na n sekund.

Ćwiczenie: Napisz program “sekundnik”, który przez określony przez użytkownika czas wyświetla liczbę sekund pozostałą do końca. Wykorzystaj pętlę for i generator range.

Moduł time

Najpotrzebniejsze funkcje:
`time.localtime()` - aktualny czas (w brzydkiej formie)
`time.time()` - aktualny timestamp (czas unixowy)

Aktualny czas w czytelnej formie stringa:

```
now = time.localtime()
time.strftime("%H : %M : %S", now)
```

Szczegóły formatowania czasu:
https://docs.python.org/3/library/time.html#time.strftime

Ćwiczenie: Rozszerz poprzedni program sekundnik o wyświetlanie aktualnej daty (DD-MM-RRRR) na starcie oraz w pętli bieżącego czasu (HH:MM:SS). Przy pomocy timestampów oblicz czas wykonania programu (end - start)

Moduł datetime

```
import datetime
```

Moduł datetime zapewnia wygodniejsze operowanie datami oraz czasem w formie obiektowej.
`datetime.datetime` - data i czas
`datetime.date` - tylko daty

np.
`datetime.datetime.now(), datetime.datetime.time()` - aktualny czas i data
`datetime.date.today()` - aktualna data
`datetime.date.today().year, datetime.date.today().month, itp.` - pola obiektu daty

Moduł datetime

Operacje na datach:

```
d1 = datetime.date(2020, 5, 18)
d2 = datetime.date(year=2020, month=5, day=25) # jawne parametry
print(d1, d2)

delta = d2 - d1
print(delta)

d3 = d2 + datetime.timedelta(days = 6)
print(d3)
```

Inne sposoby importowania modułów

- nadanie własnej nazwy: `import module as m`

Odwołanie do modułu: m.funkcja()

```
import math as majca
print(majca.sqrt(5))  #zamiast math.sqrt(5)
```

- import konkretnych submodułów (klas w modułach): `from module import submodule`

```
from datetime import date
print(date.today()) # zamiast datetime.date.today()
```

- import wszystkich nazw (w tym submodułów) z modułu: `from module import *` - wtedy wszystkie nazwy są dostępne “od ręki” bez nazwy modułu

```
from math import *
print(sqrt(5))
```

Dlaczego upraszczanie niezalecane: nie wiemy z którego modułu pochodzi użyta funkcja i przy większej liczbie modułów może dojść do konfliktu nazw

Funkcja dir

`dir()` - pokazuje jakie funkcje i nazwy są zawarte w danym module lub jakie funkcje zdefiniowane są dla danego typu (klasy)

```
import math, random, keyword

for i in math, random, keyword, list, tuple:
    print("Functions in " + str(i))
    print(dir(i))
    print("\n")
```

Inne moduły warte poznania w przyszłości

- turtle - “LOGO” w Pythonie, umożliwia graficzną pracę z żółwikiem ()
- tkinter - wbudowana prosta biblioteka do tworzenia GUI
- json, csv - wbudowane moduły do pracy z plikami JSON i CSV
- email, smtp - do wysyłania maili z poziomu skryptu
- webbrowser - do obsługi przeglądarki z poziomu skryptu
- re - do wyrażeń regularnych (regexp)
- statistics - operacje związane ze statystyką
- itertools - narzędzia do efektywnego przetwarzania iteratorów, ułatwiając operacje na sekwencjach
- calendar - operacje związane z kalendarzem (generowanie kalendarzy, obliczanie dni tygodnia)
- unittest - framework do tworzenia i uruchamiania testów jednostkowych w Pythonie
- sys - dostęp do zmiennych i funkcji związanym z interpreterem Pythona
- urllib - do pracy z URL-ami, umożliwia pobieranie zawartości stron internetowych
- pickle - umożliwia serializację obiektów Pythona, przydatne do zapisywania i odczytywania danych
- collections - zawiera dodatkowe struktury danych, rozszerzając możliwości podstawowych typów danych Pythona

Tworzymy własny moduł

plik kalkulator.py

```
def dodaj(a,b):
    return a+b
def odejmij(a,b):
    return a-b
def pomnoz(a,b):
    return a*b
```

plik main.py

```
import kalkulator

x = kalkulator.dodaj(2,4)
y = kalkulator.odejmij(1,2)
z = kalkulator.pomnoz(5,6)
print(x,y,z)
```

Uwaga! do prostego importu pliki muszą znajdować się w tej samej lokalizacji!

Więcej modułów = pakiet!

Kiedy mamy więcej plików modułów, nie trzymamy ich razem w jednej lokalizacji, tylko dzielimy logicznie na pakiety (“moduły modułów”)

Przykładowa struktura:

main.py:

- pakiet1:
  - modul1.py
  - modul2.py
- pakiet2:
  - modul21.py
  - modul22.py
  - modul23.py

Pakiety

Pakiet to folder z modułami i **plikiem o nazwie `__init__.py`**
Plik **`__init__.py`** może być nawet pusty, ale informuje, że dany folder to moduł Pythona

Przykładowy pakiet:
folder: kalkulatory
pliki: kalkulator.py , kalkulator_naukowy.py , __init__.py

- kalkulator.py ->
  proste funkcje, np. dodaj, odejmij, pomnoż itp.
- kalkulator_naukowy.py ->
  funkcje z użyciem modułu math, np. pierwiastek, potega, sinus itp.
- __init__.py -> pusty plik bez wartosci

Importowanie pakietu i modułów

w “głównym” skrypcie:

```
import kalkulatory.kalkulator
import kalkulatory.kalkulator_naukowy

x = kalkulatory.kalkulator.dodaj(1,2)
s = kalkulatory.kalkulator_naukowy.sinus(90)
print(x,s)
```

Importowanie pakietu i modułów

lub inny sposób:

```
from kalkulatory import kalkulator
from kalkulatory import kalkulator_naukowy

x = kalkulator.pomnoz(4,5)
s = kalkulator_naukowy.pierwiastek(5)
print(x,s)
```

Co robi import?

import = “include” + “run” (dołączenie modułu i uruchomienie)

Przykład:

plik kalkulator.py

```
print(“Instrukcja z modulu kalkulator”)

...
```

plik main.py

```
import kalkulator
print(“Instrukcja z pliku main”)
...
```

> Instrukcja z modulu kalkulator
> Instrukcja z pliku main

__name__

Czasem “uruchomienie” modułu jest konieczne przy imporcie, np. do sprawdzenia czy zaimportowano wcześniej inne konieczne moduły. Niekiedy umieszczamy w nim przypadki testowe, które chcemy uruchomić, kiedy moduł jest uruchamiany “samodzielnie”.

W takim przypadku często występuje w module konstrukcja:

```
if __name__ == “__main__”:

    # instrukcje jeśli moduł jest uruchamiany samodzielnie, np. testy
```

Python rozróżnia przypadki samodzielnego uruchamiania modułu (skryptu) od importowania. Przypisywana jest wtedy wartość do ukrytej zmiennej __name__ (przyjmuje wartość “__main__ “ dla samodzielnego wywołania albo nazwę modułu dla importu).

Instalowanie bibliotek

Jeśli chcemy zaimportować w Pythonie zewnętrzną bibliotekę, musimy ją najpierw zainstalować (uwaga! niektóre dystrybucje zawierają pakiet pre-instalowanych bibliotek, np. Anaconda zawiera już Numpy, Pandas, Matplotlib itp.)

Przykład:
W terminalu/cmd lub do wyklikania w IDE:

```
pip install numpy
```

Po zainstalowaniu w systemie importujemy jak “zwykły” moduł

```
import numpy as np
```

Operacje na plikach

Podstawowe operacje na plikach w Pythonie to:

- otwieranie i zamykanie (open / close)
- odczyt (read)
- zapis (write)
- dodanie zawartości (append)

Otwieranie plików tekstowych i odczyt zawartości

Aby otworzyć plik należy użyć funkcji **open()**

```
my_file = open(filename, mode)
```

gdzie:

`my_file` - zmienna "przechowująca" otwarty plik (tzw. handler)

`filename` - nazwa pliku, który jest w tym samym katalogu co skrypt lub względna/bezwzględna ścieżka do pliku

`mode` - odczyt `"r"` (domyślnie), zapis `"w"`, dodanie zawartości `"a"`

Lokalizacja pliku:

- w tym samym folderze co skrypt: np. `"nazwa.txt"`
- bezwzględna: np. `"C:\\Nowyfolder\\nazwa.txt"`

Odczyt pliku

Pliki tekstowe po otwarciu można odczytać na różne sposoby:

- w całości - funkcja `read()`
- n znaków - funkcja `read(n)`
- jedną linię - funkcja `readline()`
- wszystkie linie - funkcja `readlines()`

Odczyt pliku - przykład

Plik “demo.txt” o zawartości:

To jest pierwsza linia

To jest druga linia

A to trzecia

```
my_file = open(“demo.txt”, “r”)
content = my_file.read()
print(content)
```

Wyświetla całą zawartość pliku

Odczyt pliku - przykład

```
my_file = open(“demo.txt”, “r”)
first_ten = my_file.read(10)
next_ten = my_file.read(10)
print(first_ten)
print(next_ten)
```

> To jest pi
> erwsza lin

Wyświetla 10 znaków i przenosi “wskaźnik” za odczytany tekst

Odczyt pliku - przykład

```
my_file = open(“demo.txt”, “r”)
for i in range(3):
    print(my_file.readline())
```

```
> To jest pierwsza linia
>
> To jest druga linia
>
> A to trzecia
```

Wyświetla 3 linie pliku - ale każda linijka (poza ostatnią) w pliku ma na końcu “ukryty” znak “\n” (koniec linii), który jest wyświetlany jako pusta linia

Odczyt pliku - przykład

```
my_file = open(“demo.txt”, “r”)
tab = my_file.readlines()
print(tab)
print(tab[0])
```

```
> ['To jest pierwsza linia\n', 'To jest druga linia\n', 'A to trzecia']
> To jest pierwsza linia
>
```

Zwraca listę z kolejnymi linijkami pliku, można jej używać jak normalnej listy napisów

Zamknięcie pliku

Otwierane pliki należy zamykać po użyciu (w szczególnym przypadku bardzo duży niezamknięty plik może spowodować obciążenie pamięci)

```
my_file = open(“demo.txt”, “r”)
# operacje
# operacje
# operacje
my_file.close()
```

Słowo kluczowe with

Bardziej “pythonic” jest jednak używanie składni `with-as` do otwierania plików i operacji na nich

```
with open(“demo.txt”, “r”) as my_file:
    # operacje
    # operacje
    # operacje
```

Wtedy nie musimy pamiętać o zamykaniu pliku - zostanie automatycznie zamknięty po zakończeniu bloku `with`

Zapis do pliku

Aby zapisywać do pliku musimy go utworzyć w trybie “w” (write)
Jeśli plik nie istnieje, to zostanie utworzony. Jeśli plik istnieje, zawartość zostanie wykasowana!

```
with open("my_file.txt", "w") as f:
    f.write("To jest nowy plik\nI jego nastepna linijka")
    f.write("oraz jej ciag dalszy")
    f.write("\nDopiero teraz zaczyna sie trzecia")

with open("my_file.txt", "r") as f:
    print(f.read())
```

> To jest nowy plik
> I jego nastepna linijkaoraz jej ciag dalszy
> Dopiero teraz zaczyna sie trzecia

WAŻNE: `write()` przyjmuje jako argument tylko stringi!

Dopisywanie na końcu pliku

Aby dopisywać coś do pliku musimy go utworzyć w trybie “a” (append).
Jeśli plik nie istnieje, to zostanie utworzony.

```
with open("my_file.txt", "a") as f:
    f.write("\n")
    f.write("Dopisuje czwarta linijke!")

with open("my_file.txt", "r") as f:
    print(f.read())
```

> To jest nowy plik
> I jego nastepna linijkaoraz jej ciag dalszy
> Dopiero teraz zaczyna sie trzecia
> Dopisuje czwarta linijke

Operacje na plikach

Ćwiczenie: Stwórz tablicę 5 imion uczestników kursu. Następnie stwórz plik names.txt, w którym w każdej linijce będzie jedno imię oraz liczba jego liter, a następnie wczytaj zawartość pliku i utwórz 2 tablice: z imionami i liczbą liter (na podstawie odczytu z pliku). Wykorzystaj składnię with-as.

Wyjątki

Wyjątek - nieoczekiwany błąd powodujący przerwanie programu

Przykłady:

- użycie nieprawidłowego typu (np. string + int)
- nieprawidłowa operacja matematyczna (dzielenie przez 0!)
- nieprawidłowa operacja rzutowania (np. int(“text”))
- wyjście poza zakres tablicy
- odczyt nieistniejącego pliku (lub nieprawidłowa ścieżka)

Obsługa wyjątków: try-except

Jeśli spodziewamy się, że operacja może wywołać błąd (np. plik może nie istnieć lub użytkownik może nie wpisać tego, co oczekujemy) otaczamy instrukcje blokiem try i próbujemy obsłużyć ("złapać") wyjątek w bloku except.

```
try:
    # operacje mogące wywołać błąd
except NazwaSpodziewanegoBłędu:
    # operacje wykonywane w przypadku wystąpienia wyjątku
```

Obsługa wyjątków: try-except

Przykład:

```
try:
    print(5/2)
    print(10/5)
    print(1/0) # ta linia spowoduje blad - przejscie do except
    print(5/5) # ta linia się nie wykona - poprzednia zakonczyla blok try
except ZeroDivisionError:
    print(“Nie mozna dzielic przez zero!”)
```

> 2.5
> 2.0
> Nie mozna dzielic przez zero!

Obsługa wyjątków: try-except

Przykład otwarcia nieistniejącego pliku:

```
try:
    with open(“new_file.txt”, “r”) as f:
        print(f.read())
except FileNotFoundError:
    print(“Plik nie istnieje i nie mogl zostac otwarty!”)
except:
    print(“Dowolny inny blad”)
```

> Plik nie istnieje i nie mogl zostac otwarty!

Uwaga: używanie pustego `except` wydaje się być wygodne, ale jest niezalecane - programista powinien przewidzieć, jakie błędy mogą wystąpić i obsłużyć te konkretne!

Obsługa wyjątków: try-except-finally

Istnieje też instrukcja finally, która wykona się niezależnie od tego czy zostanie złapany wyjątek (w praktyce nie jest zbyt często używana w prostych programach - w większości przypadków podobnie zachowuje się kod po bloku try-except w którym zostanie złapany wyjątek)

```
try:
    print(10/0)
except ZeroDivisionError:
    print(“Nie mozna dzielic przez zero!”)
finally:
    print(“Instrukcja z bloku finally”)
```

> Nie mozna dzielic przez zero!
> Instrukcja z bloku finally

Obsługa wyjątków: try-except

Ćwiczenie: Poproś użytkownika o podanie liczby, używając `input()` i wyświetl jej kwadrat. Obsłuż przy użyciu bloku `try-except` sytuację, kiedy użytkownik jest złośliwy i poda napis - niech wtedy liczba ma arbitralnie wartość 0.

```
try:
    x = int(input("Podaj liczbe:"))
except ValueError:
    x = 0

print(x**2)
```

Rzucanie wyjątków

Możemy sami rzucać wyjątkami, jeśli chcemy w danych sytuacjach przerywać dalsze działanie programu (albo np. zmusić użytkownika funkcji do obsługi danego wyjątku). Służy do tego instrukcja `raise`.
Przykład:

```
def square_area(a):
    if a <= 0:
        raise ValueError
    return a*a

try:
    print(square_area(5))
    print(square_area(-5))
except ValueError:
    print("Uzytkownik nie zna matematyki")
```

Ćwiczenia interaktywne - do samodzielnego potestowania wiedzy

1) Quiz: Co to za moduł:
   https://forms.gle/4yStmRoTwiAXsqcZ9

2) Quiz: Jaki to wyjątek:
   https://forms.gle/Ah6qMET7AVtnoSPC8

3) Python Challenge - do sprawdzenia wiedzy i pracy mózgu :)
   http://www.pythonchallenge.com/

4) Code Combat - do poćwiczenia algorytmiki i jako wprowadzenie do programowania obiektowego:
   https://codecombat.com/play/dungeon?hour_of_code=true

Zadania do wykonania i przesłania po zajęciach 3

1. **(1pkt) Porządkowanie plików.**

Katalog bazowy zawiera foldery o nazwach np. "wakacje", "zima", "wyjazd" itd., które zawierają zdjęcia w formacie .jpg lub .png (rozszerzenie może być pisane dużymi lub małymi literami). Napisz skrypt, który stworzy w katalogu bazowym nowy folder "kopie", następnie wylistuje zawartość każdego folderu (tylko pliki ze zdjęciami!) i skopiuje pliki z każdego folderu do "kopie", nadając im nowe nazwy według wzorca: NAZWAFOLDERU_NR (+.jpg/.png), gdzie NR to numer porządkowy zdjęcia w folderze liczony od 0. Przygotuj też raport z operacji - zapisz w pliku txt nazwę każdego folderu z pełną ścieżką bezwzględną, a pod nim stare nazwy plików. Wykorzystaj wbudowane moduły **os** - do operacji na ścieżkach i nazwach plików oraz **shutil** - do kopiowania plików.

2. **(1pkt) Wysyłka maila.**

Napisz skrypt, który odczyta treść pliku o podanej nazwie, a następnie wyśle maila z tą treścią oraz ustaloną nazwą nadawcy i tytułem (zahardkodowane lub pobrane przez input).
Wykorzystaj wbudowane moduły **email** i **smtp** do połączenia się z serwerem i wysłania maila. Skorzystaj np. z ustawień prywatnej lub uczelnianej skrzynki, ale odpowiednio zanonimizuj dane w końcowym skrypcie - na przykład oczytując hasło z lokalnego pliku albo używając jakiegoś szyfrowania.

3. **(1pkt) Kalendarz wypłat.**

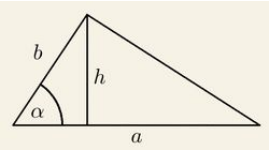
Wypłata dla pracowników firmy musi być zrobiona w ostatni dzień roboczy każdego miesiąca. Na wejściu należy podać rok, a na wyjściu mają być wyświelone miesiące (nazwy słowne po ang.) oraz data wypłaty - ostatni dzień miesiąca, który nie jest sobotą ani niedzielą, odpowiednio sformatowany do polskiego systemu zapisu dat. Uwzględnij lata przestępne.
Wykorzystaj moduły **datetime** do wyświetlania daty i **calendar** do sprawdzania dnia tygodnia.
(*dla ambitnych - można utrudnić zadanie, żeby ostatni dzień nie wypadał w święto - sprawdź zewnętrzną bibliotekę holidays: https://pypi.org/project/holidays/)

4. **(2pkt) Baza danych LEGO.**

Pobierz bazę danych zestawów LEGO ze strony: https://cdn.rebrickable.com/media/downloads/sets.csv.gz?1698044941.1218421 - w tabeli jest m.in. rocznik wydania zestawu oraz link url do zdjęcia. Wykorzystaj wbudowany moduł **csv** do oczytu pliku (w przyszłości będziemy używać do tego Pandas). Na wejściu należy podać rocznik (od 1949 do 2024), a na wyjściu mają się wyświetlić następujące dane: ilość zestawów z danego roku oraz średnia i mediana ilości elementów w zestawach w danym roku. Następnie w przeglądarce ma się wyświetlić losowe zdjęcie zestawu z tego roku. Do obliczeń wykorzystaj operacje na listach/słownikach oraz moduł **statistics**, do losowania zestawu użyj random, a do otwarcia przeglądarki z danym linkiem moduł **webbrowser**.

Zadania dodatkowe - do poćwiczenia dla chętnych

1. Zdefiniuj krotkę składającą się z 10 losowych elementów, z przedziału wpisanego przez użytkownika. Oblicz średnią geometryczną elementów krotki - pierwiastek n-tego stopnia z iloczynu wszystkich n liczb.
2. Napisz program do obliczania pola trójkąta ostrokątnego, gdzie dane są tylko liczby a, b i α (w stopniach). Sprawdź czy dane podane przez użytkownika są poprawne, czyli trójkąt jest naprawdę ostrokątny oraz spełnione są odpowiednie nierówności.



3. Napisz grę w zgadywanie liczby. Użytkownik podaje zakres liczb całkowitych, z których program losuje jedną. Następnie prosi użytkownika o zgadnięcie liczby i w przypadku błędnego wskazania wyświetla podpowiedzi "za dużo" lub "za mało". Dozwolone są 3 próby zgadnięcia. Sprawdź, czy użytkownik nie chcę oszukać - np. czy zakres nie jest zbyt mały dla tylu prób oraz czy podawane są faktycznie numery (wykorzystaj funkcję: isnumeric()).

4. Napisz program, który prosi użytkownika o podanie roku, miesiąca i dnia. Następnie wyświetla takie informacje jak:
   - dzień roku (pamiętaj, że są też lata przestępne)
   - numer tygodnia
   - datę 2 tygodnie przed i po podanej dacie
   - datę następnej niedzieli
   - czas unixowy bieżącej godziny w podanym dniu

Zadania dodatkowe – do poćwiczenia dla chętnych

1. Wygeneruj 10 losowych liczb z przedziału podanego przez użytkownika. Zapisz w pliku tekstowym w każdej linijce tyle gwiazdek, ile wynosi wartość każdej z liczb. Następnie odczytaj plik i wyświetl w każdej linijce tyle kropek ile gwiazdek było w każdej linijce (wykorzystaj listy). Następnie dopisz te kropki do końca poprzedniego pliku, ale możesz tylko dwukrotnie wywołać funkcję write. Podpowiedź: jakim znakiem połączysz elementy tablicy z kropkami?

2. Generator ocen na studiach ;-) Stwórz tablicę z nazwiskami 5 studentów. Dla każdego studenta wylosuj ocenę (ze skali na studiach: 2.0, 3.0., 3.5, 4.0., 4.5, 5.0) i zapisz wszystko w zmiennej będącej słownikiem (klucz - nazwisko, wartość - ocena). Następnie zapisz zawartość słownika w pliku tekstowym, gdzie każda linijka ma postać:
   `kolejny_numer . nazwisko : ocena`
   Uwaga: **`kolejny_numer`** możesz wyznaczyć normalnie przy pomocy zmiennej zwiększającej swoją wartość w pętli. Potem poszukaj czym jest **`enumerate()`** w Pythonie i zobacz jak może tutaj ułatwić życie.

   Dla ambitnych: spróbuj poradzić sobie z przypadkiem, gdyby jakieś nazwisko na liście się powtarzało (w liście jest to oczywiście możliwe, ale słownik nie pozwoli na zduplikowanie takiego samego klucza). Możesz natomiast nazwać drugiego Kowalskiego jako "Kowalski2", trzeciego jako "Kowalski3" itd. Może się przydać funkcja **`count()`** - poszukaj jej działania dla listy.

3. Obsłuż wyjątki, które wystąpią w przypadku podanych operacji, tak aby program nie zakończył działania a użytkownik dostał stosowny komunikat. Typ wyjątku możesz sprawdzić wywołując po prostu błędny kod. Wszystkie wyjątki Pythona opisane są m.in. w dokumentacji: https://docs.python.org/3/library/exceptions.html (jest ich dużo i nie trzeba się ich uczyć, ale kilka z nich warto znać, aby umieć rozpoznać błąd)

   a.
   ```
   my_list = [1,2,3]
   x = my_list[5]
   ```
   b.
   ```
   my_dict = {"apples": 3, "bananas":5, "oranges": 9}
   print(my_dict["cherries"])
   ```
   c.
   ```
   print("Zmyslow" + 5)
   ```
   d.
   ```
   import maths
   ```
   e.
   ```
   x = 1
   y = 2
   z = w
   ```