Laboratorium 5 (Programowanie 2 – 410-KS1-2PRO3)
Kierunek kognitywistyka i komunikacja
Funkcje rekurencyjne.
Listy składane (ang. *list comprehension*).
Obsługa wyjątków.

Funkcje rekurencyjne

1. **Rozważmy trzy następujące przykładowe kody funkcji wyznaczające sumę kolejnych liczb naturalnych:**

```
def sum(n):
    total = 0
    for index in range(n+1):
        total += index

    return total
```

```
def sum(n):
    if n > 0:
        return n + sum(n-1)
    return 0
```

```
def sum(n):
    return n + sum(n-1) if n > 0 else 0
```

Omów różnice. Zwróć uwagę na rekursję w definicjach drugiej i trzeciej oraz na operator trójargumentowy (ang. *ternarny operator*) w definicji ostatniej.

2. **Napisz funkcję (nazwa funkcji *silnia*), która będzie wyznaczała iloczyn kolejnych liczb naturalnych od 1 do zadanej liczby, przy czym dla zera funkcja ma zwracać 1. Rozwiąż zadanie korzystając z pętli oraz z funkcji rekurencyjnej.**
3. **Napisz funkcję rekurencyjną, która będzie wyznaczał *n*-ty wyraz ciągu Fibonacciego tzn. ciągu, którego dwa pierwsze wyraz są równe odpowiednio 0 i 1, a każdy kolejny jest sumą dwóch go bezpośrednio poprzedzających.**

Listy składane.

1. **Przeanalizuj następujące kody tworzące listę liczb parzystych z zakresu od 1 do 10. Wyświetl na konsoli wyniki ich działania.**

```
liczby = []
for i in range(1,10):
    liczby.append(i)
```

oraz

```
liczby = [i for i in range(1, 10)]
```

2. **Napisz funkcję, która poprosi użytkownika o podanie liczby całkowitej dodatniej, a następnie w wypisze na konsoli liczby będące kolejnymi potęgami liczby 2 z zakresu od 1 do liczby podanej przez użytkownika. Wykorzystaj raz pętle, a raz listy składane.**
3. **Przeanalizuj następujące kody tworzące listę liczb parzystych z zakresu od 1 do 10. Wyświetl na konsoli wyniki ich działania.**

```
liczby = []
for i in range(1,10):
```

```
    if i % 2 == 0:
        liczby.append(i)
```

oraz

```
liczby = [i for i in range(1, 10) if i % 2 == 0]
```

4. **Korzystając z poprzedniego przykładu napisz funkcję, która poprosi użytkownika o podanie liczby całkowitej dodatniej, a następnie w wypisze na konsoli liczby nieparzyste z zakresu od 1 do liczby podanej przez użytkownika. Wykorzystaj raz pętle, a raz listy składane.**
5. **Napisz funkcję, która poprosi użytkownika o podanie liczby całkowitej dodatniej, a następnie wypisze na konsoli liczby będące kolejnymi kwadratami liczb parzystych (nieparzystych) z zakresu od 1 do liczby podanej przez użytkownika. Wykorzystaj raz pętle, a raz listy składane.**
6. **Przeanalizuj następujący kod**

```
[z**2 if z % 2  == 0 or z % 3 == 0 else z**3 for z in range(1,10)]
```

Zapisz jego odpowiednik przy użyciu pętli. Zbuduj podobny do niego przykład. Konstruując go użyj pętli i listy składanej.

Obsługa wyjątków.

1. Hierarchia wyjątków (Python 3.9 https://docs.python.org/3.9/library/exceptions.html)[1]:

```
BaseException
 +-- SystemExit
 +-- KeyboardInterrupt
 +-- GeneratorExit
 +-- Exception
      +-- StopIteration
      +-- StopAsyncIteration
      +-- ArithmeticError
      |    +-- FloatingPointError
      |    +-- OverflowError
      |    +-- ZeroDivisionError
      +-- AssertionError
      +-- AttributeError
      +-- BufferError
      +-- EOFError
      +-- ImportError
      |    +-- ModuleNotFoundError
      +-- LookupError
      |    +-- IndexError
      |    +-- KeyError
      +-- MemoryError
      +-- NameError
      |    +-- UnboundLocalError
      +-- OSError
      |    +-- BlockingIOError
      |    +-- ChildProcessError
      |    +-- ConnectionError
      |    |    +-- BrokenPipeError
      |    |    +-- ConnectionAbortedError
      |    |    +-- ConnectionRefusedError
      |    |    +-- ConnectionResetError
      |    +-- FileExistsError
      |    +-- FileNotFoundError
      |    +-- InterruptedError
      |    +-- IsADirectoryError
      |    +-- NotADirectoryError
      |    +-- PermissionError
      |    +-- ProcessLookupError
      |    +-- TimeoutError
      +-- ReferenceError
```

[1] Hierarchia wyjątków może być inna dla innych wersji Pythona.

```
+-- RuntimeError
|    +-- NotImplementedError
|    +-- RecursionError
+-- SyntaxError
|    +-- IndentationError
|         +-- TabError
+-- SystemError
+-- TypeError
+-- ValueError
|    +-- UnicodeError
|         +-- UnicodeDecodeError
|         +-- UnicodeEncodeError
|         +-- UnicodeTranslateError
+-- Warning
     +-- DeprecationWarning
     +-- PendingDeprecationWarning
     +-- RuntimeWarning
     +-- SyntaxWarning
     +-- UserWarning
     +-- FutureWarning
     +-- ImportWarning
     +-- UnicodeWarning
     +-- BytesWarning
     +-- EncodingWarning
     +-- ResourceWarning
```

Najczęstsze rodzaje błędów:

- **ArithmeticError**: Klasa bazowa wyjątków wbudowanych wywoływanych w przypadku wystąpienia różnych błędów arytmetycznych: *OverflowError, ZeroDivisionError, FloatingPointError*
- **LookupError**: Klasa bazowa wyjątków wywoływanych w przypadku użycia niewłaściwego klucza lub indeksu dla uzyskania dostępu do elementów sekwencji: *IndexError, KeyError*
- **EOFError**: Wywoływany w przypadku, gdy jedna z funkcji wbudowanych *input()* lub *raw_input()* osiągnie koniec pliku (*EOF*).
- **FloatingPointError**: Wywoływany w przypadku błędu operacji zmiennoprzecinkowej.
- **IOError**: Wywoływany w przypadku wystąpienia błędu operacji wejścia/wyjścia instrukcji print, funkcji wbudowanej open() czy też metod obiektów plikowych.
- **ImportError**: Wywoływany w przypadku wystąpienia błędu instrukcji import.
- **IndexError**: Wywoływany w przypadku użycia indeksu spoza zasięgu.
- **KeyError**: Wywoływany w przypadku nie wystąpienia klucza.
- **MemoryError**: Wywoływany w przypadku wyczerpania się pamięci operacji.
- **NameError**: Wywoływany w przypadku nieodnalezienia lokalnej lub globalnej nazwy.
- **OverflowError**: Wywoływany w przypadku wystąpienia wyniku operacji arytmetycznej, który jest zbyt duży aby mógł być reprezentowany.
- **SyntaxError**: Wywoływany w sytuacji, gdy parser napotka błąd składniowy.
- **TypeError**: Wywoływany w sytuacji wykonania operacji wbudowanej na obiektu niewłaściwego typu.
- **ValueError**: Wywoływany w przypadku wywołania operacji z argumentem właściwego typu, lecz o nieprawidłowej wartości.
- **ZeroDivisionError**: Wywoływany w przypadku, gdy drugi argument operacji dzielenia lub reszty z dzielenia (modulo) jest równy zero.

2. Nie wykonując kodu dopasuj następujące wyjątki

ValueError
NameError
SyntaxError
KeyError
ZeroDivisionError
OverflowError

do następujących sytuacji (każdy fragment jest pełnym dostępnym kodem):

```
x * 3
```

```
x = 12
y = 12-24
x/(x+y)
```

```
x = 41
y = 123456789.12
x**y
```

```
slownik = {'pies': "Fafik",
           'kot': "Bogdan",
           'mysz': "Konrad"}
print(slownik["Bogdan"])
```

3. **Przeanalizuj następujący kod (wychwytywanie wyjątków):**

```
liczba = input("Podaj dowolną liczbę całkowitą:\n")
try:
    x = int(liczba)
    y = 10 / x
except ZeroDivisionError:
    print("Nie dzielimy przez 0!")
except:
    print("Podano niewłaściwą wartość")
else:
    print(y)
finally:
    print("Zrobione!")
```

Zastanów się czy ważna jest kolejność podawania wyjątków. Napisz kod z wyjątkami wstawiając po pierwszym *except Exception*.

4. **Przeanalizuj następujący kod (zgłaszanie wyjątków):**

```
import math

anumber = int(input('Podaj liczbę całkowitą:\n'))

if anumber < 0:
    raise RuntimeError('Nie możesz podawć liczb ujemnych!')
else:
    print(math.sqrt(anumber))
```

Napisz podobny kod.