Temat: Podział języków programowania. Narzędzia wykorzystywane do programowania.

Przeanalizuj następujące zagadnienia:
Języki programowania można podzielić na kilka różnych kategorii w zależności od różnych kryteriów. Oto kilka podstawowych podziałów:
1. Ze względu na poziom abstrakcji:
- Języki wysokiego poziomu: Bliskie naturalnemu językowi, łatwe do nauki i zrozumienia (np. Python, Java, C#).
- Języki niskiego poziomu: Bardziej zbliżone do kodu maszynowego, trudniejsze do zrozumienia i używania, ale zapewniają większą kontrolę nad sprzętem (np. Assembler, C).

2. Ze względu na paradygmat programowania:
- Programowanie proceduralne: Skupia się na sekwencji działań, które mają być wykonane (np. C, Pascal).
- Programowanie obiektowe: Umożliwia modelowanie danych w sposób zorganizowany przy użyciu obiektów (np. Java, C++, Python).
- Programowanie funkcyjne: Skupia się na używaniu funkcji i unika zmiennych stanu (np. Haskell, Scala, Erlang).
- Programowanie logiczne: Oparte na formalnej logice, np. Prolog.

3. Ze względu na zastosowanie:
- Języki ogólnego przeznaczenia: Można je stosować w różnych dziedzinach programowania (np. Python, Java, C++).
- Języki specjalistyczne: Zaprojektowane do konkretnych zadań lub obszarów (np. R do analizy statystycznej, SQL do baz danych).
- Języki skryptowe: Używane do automatyzacji zadań (np. JavaScript, Bash, Ruby).

4. Ze względu na sposób kompilacji/interpretacji:
- Języki kompilowane: Kod źródłowy jest tłumaczony na kod maszynowy przed uruchomieniem (np. C, C++).
- Języki interpretowane: Kod źródłowy jest interpretowany w czasie rzeczywistym przez interpreter (np. Python, JavaScript).

5. Ze względu na typowanie:
- Języki statycznie typowane: Typy zmiennych są określane w czasie kompilacji (np. C, Java).
- Języki dynamicznie typowane: Typy zmiennych są określane w czasie wykonywania programu (np. Python, Ruby).

6. Ze względu na platformę:
- Języki natywne: Zostały zaprojektowane do pracy na określonym systemie operacyjnym (np. C dla systemu Linux).
- Języki niezależne od platformy: Mogą działać na różnych platformach bez potrzeby modyfikacji kodu (np. Java z maszyną wirtualną).

Każdy z tych podziałów może pomóc w zrozumieniu różnorodności i specyfiki języków programowania, a także w wyborze odpowiedniego języka do określonych zadań.

Aplet w informatyce to mała aplikacja, która jest zazwyczaj uruchamiana w kontekście większego programu lub środowiska, często w przeglądarce internetowej. Aplety są najczęściej pisane w języku Java, chociaż mogą również być implementowane w innych językach.

Aplety są zaprojektowane, aby wykonywać określone zadania, takie jak wyświetlanie animacji, interakcji z użytkownikiem lub przetwarzania danych. W przeciwieństwie do tradycyjnych aplikacji, aplety nie są programami samodzielnymi i wymagają środowiska, w którym mogą działać, na przykład wtyczki przeglądarki.

Warto zauważyć, że użycie apletów spadło w ostatnich latach, szczególnie po zakończeniu wsparcia dla technologii Java w przeglądarkach, co sprawiło, że wiele aplikacji internetowych przeszło na inne technologie, takie jak HTML5 i JavaScript.

IDE (**Integrated Development Environment**) to zintegrowane środowisko programistyczne, które oferuje programistom zestaw narzędzi do pisania, testowania i debugowania kodu w jednym miejscu. IDE znacznie

ułatwia proces tworzenia oprogramowania, integrując różne funkcje, które są często dostępne w osobnych aplikacjach.

Oto kluczowe cechy IDE:
- Edytor kodu: IDE oferuje zaawansowany edytor kodu, który często wspiera funkcje takie jak podświetlanie składni, autouzupełnianie, automatyczne wcięcia i inne, co poprawia wygodę pisania kodu.
- Kompilator/Interpreter: IDE zazwyczaj zawiera kompilator lub interpreter, który umożliwia uruchamianie kodu bez konieczności korzystania z zewnętrznych narzędzi. Dzięki temu, programista może szybko sprawdzić, czy jego kod działa poprawnie. Kompilator tłumaczy kod źródłowy programu na plik wykonywalny.
- Debugger: Narzędzia debugujące pozwalają programistom na śledzenie wykonywania kodu, ustawianie punktów przerwania, analizowanie wartości zmiennych w czasie rzeczywistym i rozwiązywanie problemów.
- Zarządzanie projektami: IDE często posiada funkcje do zarządzania projektami, takie jak organizacja folderów, plików oraz możliwość łatwego dodawania nowych plików i bibliotek.
- Współpraca z systemem kontroli wersji: Wiele IDE integruje się z systemami kontroli wersji, co ułatwia zarządzanie zmianami w kodzie i wspólną pracę w zespołach.
- Pluginy i rozszerzenia: IDE oferują możliwość instalacji dodatkowych wtyczek, które mogą rozszerzać funkcjonalność środowiska i dostosowywać je do specyficznych potrzeb programisty.
- Interfejs graficzny: Dzięki graficznemu interfejsowi użytkownika, IDE są często bardziej przystępne dla nowicjuszy w porównaniu do korzystania z prostych edytorów tekstowych i terminali.

IDE znacząco przyspiesza proces programowania, poprawiając zarówno produktywność, jak i jakość samego kodu.

Przykłady popularnych środowisk IDE:
- PyCharm: IDE zaprojektowane z myślą o programistach Pythona.
- Visual Studio: IDE od Microsoftu, szczególnie popularne wśród programistów C# i .NET.
- Dev-C++ - darmowe środowisko programistyczne C/C++.
- Code::Blocks - darmowe środowisko programistyczne C/C++.
- Eclipse - otwarte środowisko programistyczne, szeroko stosowane w rozwijaniu aplikacji Java.
- IntelliJ IDEA: IDE dla języka Java, znane z zaawansowanych funkcji autouzupełniania i analizy kodu.
- Xcode: IDE dedykowane dla deweloperów aplikacji na systemy Apple.

Python to wszechstronny język programowania, który jest szeroko stosowany w różnych dziedzinach, takich jak web development, analiza danych, sztuczna inteligencja, automatyzacja oraz w wielu innych zastosowaniach. Oto kilka kluczowych cech i informacji na temat Pythona:
1. **Prosta składnia**: Python charakteryzuje się przejrzystą i prostą składnią, co sprawia, że jest łatwy do nauki, zwłaszcza dla początkujących programistów.
2. **Interaktywność**: Python oferuje interaktywny tryb, w którym można pisać i uruchamiać kod w locie, co ułatwia testowanie nowych pomysłów.
3. **Wsparcie dla wielu paradygmatów**: Python wspiera różne paradygmaty programowania, w tym programowanie obiektowe, funkcyjne i imperatywne.
4. **Bogata biblioteka standardowa**: Python ma bogaty zestaw bibliotek standardowych, które wspierają różnorodne zadania, od manipulacji danymi po rozwój aplikacji internetowych.
5. **Społeczność i wsparcie**: Python ma dużą i aktywną społeczność, co oznacza, że można łatwo znaleźć pomoc, dokumentację i zasoby.
6. **Zastosowania**: Python jest używany w wielu dziedzinach, w tym w:
   - **Web development** (frameworki Django i Flask).
   - **Analiza danych i wizualizacja** (biblioteki Pandas, NumPy, Matplotlib).
   - **Uczenie maszynowe i AI** (biblioteki TensorFlow, Keras, scikit-learn).
   - **Automatyzacja i skrypty** (programy do przetwarzania danych).
   - **Zastosowania naukowe** (biblioteki SciPy, SymPy).
7. **Aktualizacje i wersje**: Python jest stale rozwijany, a najnowszą stabilną wersją jest Python 3.12, który wprowadza nowe funkcje i poprawę wydajności w stosunku do poprzednich wersji.

Temat: Instalacja i konfiguracja środowiska programowania.

Oprogramowanie python jest domyślnie dostępne w systemie Ubuntu Live.
Wersję programu możemy sprawdzić wpisując polecenia:

```
python
python3
```

Zakończenie programu uzyskamy poleceniem:

```
exit()     #Ctrl + z
```

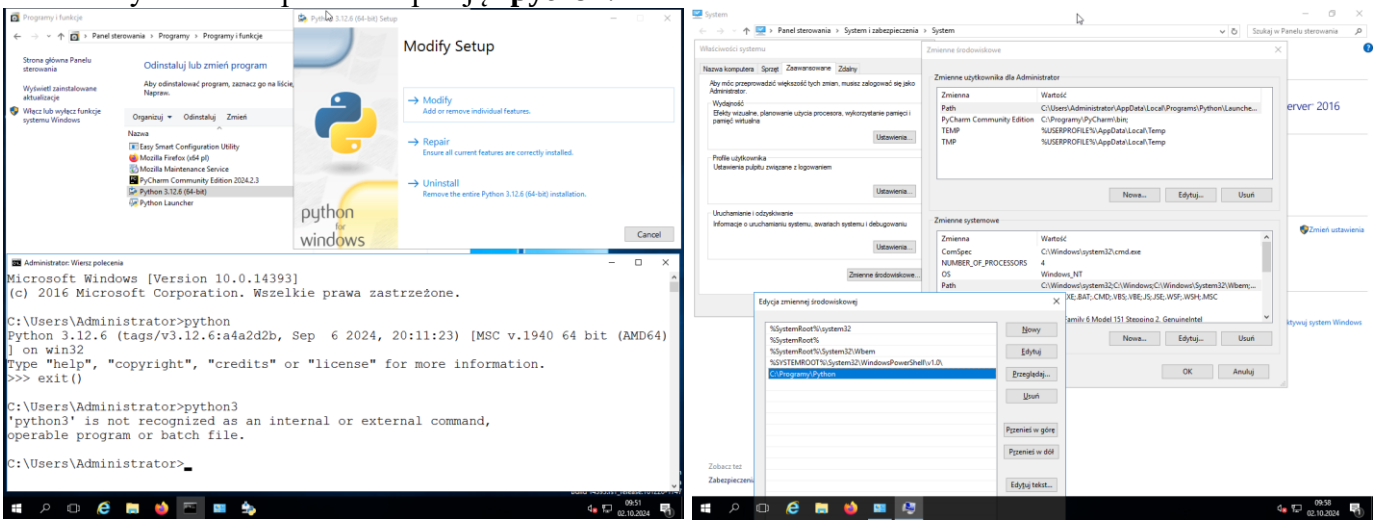
Przydatny w programowaniu będzie edytor tekstu **Geany**. Aby go zainstalować w systemie Ubuntu należy włączyć repozytoria Universe w narzędziu Oprogramowanie i aktualizacje, następnie wydać polecenie:

```
sudo adp-get install geany
```

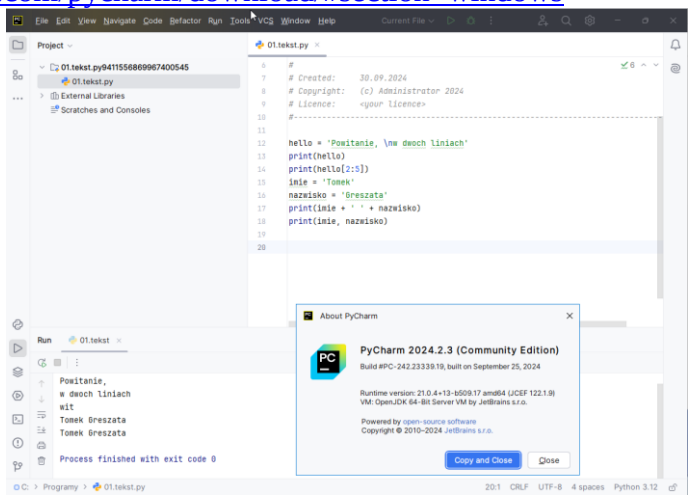
Edytor tekstu Geany umożliwia bezpośrednie wykonanie programu wybierając z menu Zbuduj => Wykonaj (F5).

Źródło **Pythona**: https://www.python.org/downloads
https://www.python.org/ftp/python/3.12.0/python-3.12.0rc3-amd64.exe

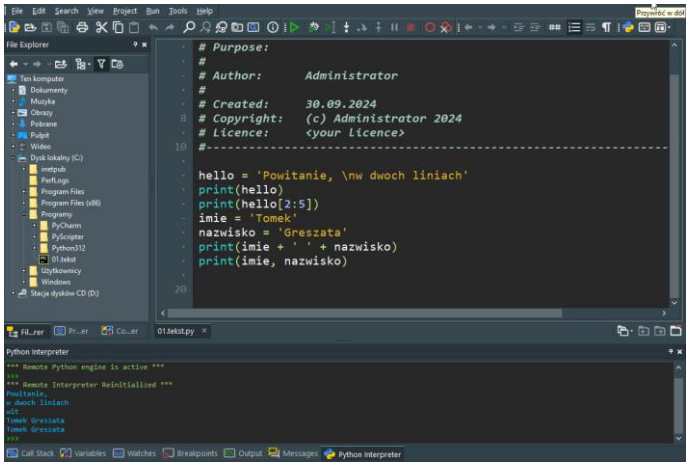
Uwaga!
Po zainstalowaniu Pythona podać ścieżkę do katalogu programu w zmiennej systemowej PATH. Program uruchamiamy z wiersza poleceń wpisując **python**.



Źródło środowiska programistycznego **PyCharm**:
https://www.jetbrains.com/pycharm/download/#section=windows



Pobrać środowisko programistyczne **PyScripter** z serwisu dobreprogramy.pl (wersja portable) lub:
https://sourceforge.net/projects/pyscripter/files/PyScripter-v5.0/PyScripter-5.0.0-x64.zip/download



Wykonaj zadanie:

Zainstaluj oprogramowanie **Python**, **PyCharm** oraz **PyScripter**. Po instalacji wykonaj zrzuty ekranowe działania tych programów i zapisz je w pliku pod nazwą **$nazwisko_$klasa_$gr_python_install** i prześlij do nauczyciela w postaci załącznika na adres greszata@zs9elektronik.pl.
Na ocenę wpływ będzie miała nazwa pliku oraz terminowość przesłania pracy.

Temat: Struktura prostego programu w języku Python.

Przydatne linki:

https://pl.wikibooks.org/wiki/Zanurkuj_w_Pythonie
https://rk.edu.pl/pl/wprowadzenie-do-pythona/

Przydatne skróty klawiszowe PyCharm:

```
ctrl + lewy przycisk myszy (w ustawieniach => general)
ctrl + / => komentowanie linii
shift + f10 => uruchomienie programu
4 spacje jako wcięcie w programie (jeden tabulator)
shift + tab => cofanie ostatniego wcięcia w programie
```

Błędy kompilatora Phyton:

- `list index out of range`: element listy z poza zakresu (błąd indeksu listy),
- `name 'x' is not definied`:
- `Can't convert 'int' object to str implicitly`: problem z interpretacją zmiennej,
- `invalid syntax`: nieprawidłowy kod programu (błąd składni),
- `name 'mesage' is not definied`: błąd nazwy zmiennej,
- `expected an indented block`: błąd wcięcia (brak wcięcia rozpoczynającego blok),
- `unexpected indent`: błąd nieoczekiwanego wcięcia,
- `'tuple' object does not supported item assignment`: błąd typu, nie można zmienić wartości,
- `EOL while scanning string literal`: błąd braku zakończenia ciągu tekstowego (brak znaku ' lub "),
- `'>=' not supported between instances of 'str' and 'int'`: błąd typów (ciąg znaków nie może być porównany z liczbą),
- `unsupported operand type(s) for %`: 'NoneType' and 'int'
- `can only concatenate str (not "int") to str`: błąd podczas wydruku ciągu znaków wraz z liczbą,
- `greet_user() missing 1 required positional argument`: 'nazwisko': błąd wywołania funkcji bez podania wartości potrzebnego argumentu funkcji,
- `non-default argument follows default argument`: błąd umieszczeniu wartości domyślnej argumentu przed argumentem niezdefiniowanym podczas wywołania funkcji.

Styl tworzonego kodu programu: https://www.python.org/dev/peps/pep-0008/

- specyfikacja PEP 8 jest najstarszym dokumentem opisującym styl Pythona,
- wcięcia wykonujemy z użyciem 4 znaków spacji (unikamy tabulatorów),
- długość wiersza nie powinna być większa od 79 znaków (ułatwia to czytanie kodu z kilku plików jednocześnie),
- nie nadużywamy stosowania pustych wierszy, ale oddzielamy nimi bloki programu,
- wokół operatorów porównania powinniśmy stosować spacje, np. `'if wiek <= 12'`,
- edytory tekstu możemy dostosować do powyższych wymagań poprzez ustawienia.

Zen Pythona, czyli zbiór reguł dotyczących tworzenia dobrego kodu w Pythonie można wyświetlić w konsoli programu wydając polecenie:

```
>>>
>>>import this
The Zen of Python, by Tim Peters

Beautiful is better then ugly.
...
>>>
```

Wiersze programu rozpoczynające się od znaku hash (#) traktowane są jako komentarze i nie wpływają na działanie programu. Komentarze można również umieszczać od dowolnego miejsca danej linii. Komentarze stosuje się do wyjaśnienia przeznaczenia i sposobu działanie danego fragmentu kodu.

```
#-----------------------------------
# Wyjaśnienie dodane przez programistę
#-----------------------------------
```

W edytorze Geany pisanie programu możemy rozpocząć z gotowego szablonu. W tym celu wybieramy z menu Plik => Nowy (z szablonu) => main.py. Plik będzie posiadał strukturę programu python (deskryptor, kodowanie znaków, funkcję main()).

```
#!C:\programy\python
#!/usr/bin/python             #deskryptor #c:\programy\python
#-*- coding: utf-8 -*-        #kodowanie znaków (utf-8, iso-8859-2 lub windows-1250)
#
def main():                   #definicja funkcji głównej main()
    return 0

if __name__ == '__main__':
    main()
#
```

Pierwszy program:

```
# Pierwszy program w pythonie: (linie komentarzy)
#print(objects, sep=' ', end='\n') #skłania funkcji print
print('Witaj w świecie Pythona!')
#
```

Pliki z programami w języku Pythona powinny posiadać rozszerzenie ***.py**. Wykonanie programu z pliku źródłowego z konsoli tekstowej uzyskamy poleceniem (wcześniej zdefiniować zmienną PATH):

```
python nazwa_pliku.py
python nazwa_pliku.py > wynik.txt
python nazwa_pliku.py < dane.txt
```

Plik z kodem źródłowym programu możemy uruchamiać w środowisku Pythona poleceniem:

```
exec(open('nazwa_pliku.py').read())
```

Kompilację pliku źródłowego programu uzyskamy poleceniem (otrzymamy plik z rozszerzeniem ***.pyc**):

```
python –m py_compile nazwa_pliku.py nazwa_pliku.pyc
```

Plik *.pyc możemy uruchamiać z menedżera plików systemu operacyjnego klikając dwukrotnie jego nazwę. Tutaj warto zastosować wstrzymanie działania programu na wskazaną liczbę sekund:

```
#wstrzymanie działania programu na podaną liczbę sekund
import time       #wczytujemy moduł time
time.sleep(5)     #na końcu programu wywołujemy wstrzymanie
```

Lub dla wstrzymania programu zastosować funkcję input:

```
input()           #zatrzymanie do wciśnięcia klawisza enter
```

Importowanie skryptu programu w celu jego uruchomienia:

```
import plik       #wczytujemy plik.py jako moduł, skrypt zostanie uruchomiony
```

Zamrożone pliki binarne: py2exe, freeze, PyInstaller

Temat: Instrukcje wejścia i wyjścia w języku Python.

Ciągi tekstowe stanowią serie znaków ujęte w znaki cudzysłowia lub apostrofu, np:

```
"Przykładowy ciąg znaków."
'Inny 2 przykładowy ciąg tekstowy.'
```

Przykłady instrukcji wyjścia (drukowania) https://docs.python.org/3/tutorial/introduction.html#text :

```
print("Jeszcze jeden komunikat")
print('Drukowanie linii \npotem kolejnej.')
print('Drukowanie linii \tpotem znak tabulatora.')
```

Dodawanie białych znaków do ciągów tekstowych:

```
# dadanie znaku tabulacji:
  print("\tWitaj!")
# wstawianie znaku końca linii:
  print("Języki programowania:\nPython\nCPP\nJava")
# wstawianie znaku końca linii i tabulacji:
  print("Języki programowania:\n\tPython\n\tCPP\n\tJava")
#\v - tabulator pionowy, \r - powrót karetki
  print('Witaj!', end=' ') #wydruk na końcu linii znaku spacji, bez przejścia do nowego wiersza
  print('Świecie!')
#
```

Usuwanie białych znaków z ciągów tekstowych, np. usunięcie spacji z wartości zmiennej:

```
#
tekst = " Jan "
print(tekst * 5)              #powtarzanie
print(tekst + tekst)          #konkatenacja
print("Usunięcie spacji z prawej strony: " + tekst.rstrip())
print("z lewej: " + tekst.lstrip())
print("z obu stron: " + tekst.strip())
#
```

Operacje na ciągach tekstowych:

```
hello = 'Witaj w świecie Pythona'
print(hello[0])         #wydrukuje pierwszy znak w łańcuchu
print(hello[2:5])       #wydrukuje znaki od 2 do 5
print(hello[:5])        #wydrukuje znaki od 0 do 5
print(hello[6:])        #wydrukuje znaki od 6 do ostatniego

imie = 'Tomasz'
nazwisko = 'Greszata'
print(imie, nazwisko)         #przecinek spowoduje oddzielenie zmiennych spacją
print(imie + nazwisko)        #zmienne będą wyświetlone bez separatora
print(imie + ' ' + nazwisko)
print(f'Witaj, {imie}')

hello = 'Witaj'
world = 'w swiecie Pythona'
print(f'{hello} {world}')
print('%s %s' % (hello, world))

tekst = "Tekst z 'cytatem' w tekście"
print(tekst)
tekst = 'Zmiana wartości zmiennej tekst'
print(tekst)

name = 'JAN kowalSKI'
print('Duże pierwsze litery: ', name.title())
print('Duże wszystkie litery: ', name.upper())
print('Male wszystkie litery: ', name.lower())
print('Kapitaliki, z dużej pierwszy znak: ', name.capitalize())
```

Łączenie ciągów tekstowych:

```
#
```

```
first_name = "jan"
last_name = "kowalski"
full_name = first_name + " " + last_name
print(full_name)
print("Witaj, " + full_name.title() + "!")
#

#łączenie napisów
a = 2
b = 4
wynik = a/b
poczatek = "Wynik dzielenia wynosi:"
koniec = "co było oczekiwane"
# %s, %d i %f to odpowiednio - napis, liczba całkowita i liczba zmiennoprzecinkowa
# Znaki te wstawione w napis zostaną zamienione wartościami zmiennych podanych
# na końcu wiersza ( % (zmienna, zmienna, itd.)
print ("%s %f %s" % (poczatek, wynik, koniec))
#
```

Pobieranie danych od użytkownika:

```
#funkcja input() wyświetli okno dialogowe
imie = input('Podaj swoje imię: ')
print(imie)
```

Temat: Typy zmiennych w języku Python.

W języku programowania Python istnieje 35 słów kluczowych, które są zarezerwowane i mają specjalne znaczenie. Oto one:

False	await	else	import	pass
None	break	except	in	raise
True	class	finally	is	return
and	continue	for	lambda	try
as	def	from	nonlocal	while
assert	del	global	not	with
async	elif	if	or	yield

Słowa kluczowe są integralną częścią składni Pythona i nie mogą być używane jako identyfikatory (nazwy zmiennych, funkcji itd.).

Reguły nazewnictwa zmiennych w Pythonie:
- nazwy powinny zawierać jedynie litery, cyfry i znaki podkreślenia, ale np. nie można zastosować nazwy zaczynającej się od cyfry: "1_zmianna",
- niedozwolone jest stosowanie znaków spacji w nazwach zmiennych,
- nie można stosować słów kluczowych Pythona, np. "print",
- nazwa zmiennej powinna być krótka i czytelna, np. lepiej "imie" niż "i",
- należy ostrożnie stosować litery l i O, ponieważ mogą pomylić się z cyframi 1 i 0.

W Pythonie istnieje wiele typów zmiennych, które są podstawą programowania w tym języku. Każda zmienna posiada unikatową nazwę oraz wartość zapisaną pod konkretnym adresem w pamięci komputera. Oto niektóre z najważniejszych typów zmiennych:

1. **Typy numeryczne**:
   - **int**: liczby całkowite, np. `5`, `-10`.
   - **float**: liczby zmiennoprzecinkowe, np. `3.14`, `-0.001`.
   - **complex**: liczby zespolone, np. `3 + 4j`.
2. **Typy sekwencyjne**:
   - **str**: ciągi znaków, np. `"Hello, World!"`.
   - **list**: lista, np. `[1, 2, 3]`, może zawierać różne typy danych.
   - **tuple**: krotka, np. `(1, 2, 3)`, jest niemutowalna.
   - **range**: przedział liczb, np. `range(0, 10)`.
3. **Typy zbiorów**:

   - **set**: zbiór, np. `{1, 2, 3}`, nie zawiera duplikatów.
   - **frozenset**: niemutowalny zbiór, np. `frozenset([1, 2, 3])`.
4. **Typy mapujące**:
   - **dict**: słownik, np. `{'key': 'value', 'another_key': 42}`, przechowuje pary klucz-wartość.
5. **Typy logiczne**:
   - **bool**: wartość logiczna, może przyjmować `True` lub `False`.
6. **None**:
   - **NoneType**: typ zmiennej o wartości `None`, która oznacza brak wartości.

Każdy z tych typów ma swoje właściwości i metody, które umożliwiają ich modyfikację i manipulację. Python jest językiem **dynamicznie typowanym**, co oznacza, że nie trzeba deklarować typu zmiennej przed jej użyciem. **Typ zmiennej jest określany automatycznie na podstawie wartości, którą przechowuje.**

Przykład zastosowania zmiennej w programie Pythona:

```
#\\   \'    \"    #metody na zachowanie znaku \
tekst = 'Jakis tekst i "cytat" w tekscie'
print(tekst)
#
q = ['element1', 'element2']  #lista
x = 'tekst'                   #łańcuch znaków   (str)
y = 100                       #liczba całkowita (int)
z = 2.13                      #liczba zmiennoprzecinkowa (float)
print(q, x, y, z)
print('%s %s %d %3.2f' % (q, x, y, z))    #%3.2f – precyzja wyświetlania
print(f'{q} {x} {y} {z}')

a == b  # a jest równe b
x != y  # x nie jest równe y
c < > d # c jest mniejsze, większe niż d
```

Można sprawdzić zachowanie się programu w przypadku pomyłek w nazwach zmiennych:

```
#
tekst = 'Jakiś tekst "z cytatem" !'
print(tekst)
#

#
tekst = "Jakiś tekst 'z cytatem' !"
print(teskt)
#
```

Znaki wprowadzane z klawiatury mają swoje odpowiedniki w tablicy znaków dla danego języka (kodowanie np. iso-8859-2). Podstawowe kodowanie ASCII zawiera tablicę 8 bitową, czyli 256 znaków, znaki języka angielskiego o numerach od 0 do 127, oraz od 128 do 255 znaki narodowe. Przy kodowaniu unikod (utf-8) tablica znaków ma rozmiar 4 bajtów. Do wyświetlenia numeru znaku używamy funkcji ord(), a do wyświetlenia znaku o określonym numerze funkcji chr():

```
ord('ż')
ord(u'ś')
chr(192)
unichr(261)
#
#
```

Python umożliwia sprawdzenie wartości zmiennych systemowych:

```
import sys
sys.platform
sys.path
```

Python umożliwia wczytanie modułu z funkcjami matematycznymi:

```
import math
math.pi                 #wartość PI
math.sqrt(15)           #pierwiastkowanie
```

```
print(math.pi)
print('%.2f' % math.pi) #precyzja wyświetlania liczby
print(math.log2(8))     #logarytm
print(math.sqrt(15))
print(min(2, 4))
print(dir(math))        #wyświetlenie dostępnych funkcji

print(pow(5,2))         #potęgowanie
round(10.6)             #zaokrąglanie
hex(10), oct(10), bin(10)     #konwersja między systemami liczbowymi
print(bin(100))
print(oct(100))
print(hex(100))
```

Pobieranie różnych typów danych od użytkownika:

```
imie = input('Podaj swoje imię: ')
print(imie)

#int(input()) określenie typu wczytywanej zmiennej (liczba całkowita)
a = int(input('Podaj liczbę całkowitą: '))
print(a * a)
print("Obwod wynosi: %d" % (4 * a ))

b = float(input('Podaj liczbę ułamkową: ')
print(b / a)
print("Iloraz liczb wynosi: %f" % (a / b))

int('5'), str(5)        #konwersja typów danych
```

Temat: Operatory matematyczne oraz logiczne.

Operacje matematyczne na liczbach całkowitych:

```
a = 2
b = 5
print(a + b)
print(a - b)
print(a * b)
print(a / b)
print(a // b)    #dzielenie całkowite
print(a % b)     #dzielenie modulo (reszta z dzielenia)
print(2 ** 16)   #potęgowanie
```

Operacje matematyczne na liczbach zmiennoprzecinkowych:

```
a = 2.0
b = 5.0
print(a + b)
print(a - b)
print(a * b)
print(a / b)
print(a % b)
```

Działania na liczba zmiennoprzecinkowych:

```
c = -4.8
print(int(c))
print(abs(c))
```

Podczas działań matematycznych obowiązuje standardowa kolejność, którą można zmienić stosując nawiasy:

```
2 + 3 * 4         # => 14
(2 + 3) * 4       # => 20
```

Przypadki specjalne – kilka instrukcji w jednym wierszu:

```
x = 1; y = 2; print(x + y)    #kilka prostych instrukcji w jednym wierszu
print(len(str(2 ** 128)))     # ilość cyfr w danej liczbie
```

Przypadki specjalne – obsługa błędów:

```
try:
    #instrukcje, które próbujemy wykonać
    print("Jakiś kod, który może sprawiać problem.")
except ZeroDivisionError:
    #instrukcje wykonywane, gdy wystąpi błąd dzelenia przez zero
    print("Nie można dzielić przez zero!")
except ValueError:
    #instrukcje wykonywane, gdy wystąpi błąd danych
    print("Nieprawidłowe dane. Wprowadź prawidłową liczbę.")
except Exception as blad:
    #identyfikacja innego błędu
    print(f"Wystąpił błąd: {blad}")
except:
    #instrukcje wykonywane, gdy wystąpi błąd inny niż powyżej wymienione
    print('Blok wykonywany opcjonalnie.')
else:
    #instrukcje wykonywane, gdy nie wystąpi żaden wyjątek
    print("Wykonaj, jeżeli nie ma wyjątku.")
finally:
    #instrukcje zamknięcia, zawsze wykonywane
    print('Blok zawsze wykonywany.')
#
```

Wykonaj zadanie:

Napisz program obliczający pole powierzchni i obwód prostokąta o długościach boków 3 i 4.

```
#
a, b = 3, 4

print("Pole powierzchni prostokąta wynosi: ")
print(a * b)

print("Obwod prostokąta wynosi: %d" % (2 * (a + b)))
input('Wciśnij klawisz enter dla zakończenia programu.')
#
```

Wykonaj zadanie:

Napisz program obliczający pole powierzchni i obwód prostokąta o długościach boków podanych przez użytkownika.

```
#
import time
print('Program obliczający pole powierzchni i obwód prostokąta.')

a = float(input('Podaj dlugosc pierwszego boku: '))
b = float(input('Podaj dlugosc drugiego boku: '))

print("Pole powierzchni prostokąta wynosi: ")
print(a * b)

print("Obwod prostokąta wynosi: %f" % (2 * (a + b)))

time.sleep(3)
#
```

Temat: Instrukcja warunkowa if.

Ogólna postać instrukcji warunkowej if:

```
if <test1>:
    <instrukcje1>       #wcięcia są bardzo istotne, pełnią funkcję dzielenia bloków
elif <test2>:           #test opcjonalny
    <instrukcje2>
else:                   #alternatywa opcjonalna
    <instrukcje3>
```

Przykłady zastosowania instrukcji if:

```
if 1: print('prawda')   #1 to wartość logiczna True
```

```
#
#
if not 1:
    print('prawda')
else:
    print('fałsz')
#
#
# a = b = c = d = 4     #inna forma zapisu
a, b, c, d = 4, 4, 4, 4
if a == 4 and b == 4 and c == 4 and \
    d == 4:
    print('wszystkie liczby są równe 4')
#
#
imie = 'Tomek'
if not "f" in imie:
    print('Brak litery f')
#
#
wiek = int(input('Podaj swój wiek: '))
if int(wiek) >= 18:
    print('Jesteś pełnoletni.')
else:
    print('Nie jestes pelnoletni.')
#
#
liczba = input('Podaj liczbę całkowitą: ')
if int(liczba) % 2 == 0:
    print('Podana liczba ' + liczba + ' jest parzysta.')
else:
    print('Podana liczba ' + liczba + ' jest nieparzysta.')
#
#
#Kilkukrotne wywołanie instrukcji elif z pominięciem bloku else.
#Zalecane, aby uniknąć błędów wynikających np. z podania nieprawidłowych danych
print('Cena biletu jest zależna od wieku.')
wiek = wiek('Podaj ile masz lat: ')
if wiek < 4:
    cena = 0
elif wiek < 18:
    cena = 5
elif wiek < 65:
    cena = 10
elif wiek >= 65:
    cena = 5
print('Cena biletu wstępu wynosi: ' + str(cena) + ' zł.')
#
#
```

Wykonaj zadanie:

Napisz program sprawdzający, która z dwóch liczb podanych przez użytkownika jest większa.

```
#
#
print('Program sprawdzajacy, ktora z podanych liczb jest wieksza.')
a = int(input('Podaj pierwsza liczbe: '))
b = int(input('Podaj druga liczbe: '))
if a > b:
    print('Pierwsza liczb jest wieksza.')
elif b > a:
    print('Druga liczba jest wieksza')
else:
    print('Liczby sa rowne.')
print('Koniec programu.')
#
#
```

Wykonaj zadanie:

Napisz program sprawdzający, czy z podanych trzech długości odcinków można zbudować trójkąt. Pracę zapisz w pliku pod nazwą **$nazwisko_$klasa_$gr_trojkat.py** i prześlij do nauczyciela w postaci załącznika na adres greszata@zs9elektronik.pl.

```
#
#
print('Program sprawdzający, czy z podanych trzech długości odcinków \
    można zbudować trójkąt.')
a = float(input('Podaj długość pierwszego boku: '))
b = float(input('Podaj długość drugiego boku: '))
c = float(input('Podaj długość trzeciego boku: '))
if a + b > c and b + c > a and c + a > b:
    print('Z podanych odcinków można zbudować trójkąt.')
else:
    print('Z podanych odcinków nie można zbudować trójkąta.')
print('Koniec programu.')
#
#
```

Temat: Pętla programowa while.

Instrukcja while jest uniwersalną pętlą w języku Python. Pętla while powtarza wykonywanie bloków instrukcji (normalnie wciętych) dopóki test znajdujący się w nagłówku będzie spełniony (zwraca prawdę).
Ogólna postać pętli while:

```
while <test1>:
    <instrukcje1>
else:                   #alternatywa opcjonalna
    <instrukcje2>       #instrukcje wykonane, jeżeli nie było break
```

Przykłady zastosowania pętli while:

```
while True: pass        #pętla pusta, zatrzymanie poprzez Ctrl + c
#
#
while True:
    print('Aby zakończyć wciśnij Ctrl + c')
#    pass         #aktywne oczekiwanie na przerwanie z klawiatury (Ctrl + c)
#
#
x = 'Tomek'
while x:                #póki x nie jest puste
    print(x, end=' ')
    x = x[1:]
#
#
x = 'Tomek'
while x:                #póki x nie jest puste
    print(x, end=' ')
    x = x[:-1]
#
#
a = 0, b = 10
while a < b:
    print(a, end=' ')
    a += 1
#
#
i = 1
while i <= 5:
    print('licznik = %s' % i)
    i += 1
#
#
i = 5
while i > 0:
    print('licznik =', i)
    i -= 1
#
```

```
#
print('Sumowanie liczb z listy')
lista = [1, 2, 3, 4, 5]
suma = 0
while lista:
    suma += lista[0]
    lista = lista[1:]
print(suma)
#
#
prompt = "\nNapisz cos o sobie, co zotanie wyswietlone na ekranie."
prompt += "\nWyraz 'koniec' zakonczy dzialanie programu."
message = ''
while message != 'koniec':
    message = input(prompt)
    if message != 'koniec':
        print(message)
#
#
```

Ogólny format zatrzymania pętli while:

```
while <test1>:
    <instrukcje1>
    if <test2>: break         #wyjście z pętli
    if <test3>: continue      #przechodzi do początku pętli
else:
    <instrukcje2>
```

Przykłady zastosowania pętli while z instrukcjami braek i continue:

```
x = 10
while x:
    x = x - 1
    if x % 2 != 0: continue
    print(x, end=' ')
#
#
x = 10
while x:
    x = x - 1
    if x % 2 == 0:
        print(x, end=' ')
#
#
print('"stop" kończy wprowadzanie danych')
while True:
    powtorz = input('Podaj tekst: ')
    if powtorz = 'stop':
        break
    elif not powtorz.isdigit():
        print('Błąd danych!' * 5)
    else:
        print(int(powtorz) ** 2)
print('Koniec')
#
#Sprawdzanie błędów
print('"stop" kończy wprowadzanie danych')
while True:
    powtorz = input('Podaj tekst: ')
    if powtorz == 'stop': break
    try:                                #wykonane podstawowo, jako ryzykowne
        num = int(powtorz)
    except:                             #wykonane, gdy wystąpi błąd
        print('Błąd danych!' * 5)
    else:                               #wykonane, gdy nie było błędu i brake
        print(int(powtorz) ** 2)
print('Koniec')
#
```

```
#
print('"stop" kończy wprowadzanie danych')
while True:
    name = input('Podaj swoje imie: ')
    if name = 'stop': break
    print('Witaj, ', name)
#
#
print('Program sprawdzający, czy dana liczba jest liczbą pierwszą.')
y = int(input('Podaj liczbę całkowita: '))
x = y // 2      #dla y > 0
while x > 1:
    if y % x == 0:
        print(y, ' ma dzielnik', x)
        break
    x -= 1
else:
    print(y, ' jest liczba pierwsza.')
print('Koniec programu.')#
#
# bin(10) - funkcja Pythona
print('Program zamienia liczbę dziesiętną na dwójkową.')
d = int(input('Podaj liczbę do konwersji: '))
b = ''
while d > 0:
    b = str(d % 2) + b
    d = d // 2
print(b)
#
#
```

Wykonaj zadanie:

Napisz program wyświetlający tabliczkę mnożenia dla liczb od 1 do 10.

```
#
print('Program wyświetlający tabliczkę mnożenia dla liczb od 1 do 10.')
i = 1
while i <= 10:
    j = 1
    while j <= 10:
        print(i * j, end='\t')
        j += 1
    print()
    i += 1
print('Koniec programu.')
#
#
```

Wykonaj zadanie:

Napisz program z pętlą while obliczający sumę tylu liczb i takich, jakie poda użytkownik. Pracę zapisz w pliku pod nazwą **$nazwisko_$klasa_$gr_suma.py** i prześlij do nauczyciela w postaci załącznika na adres greszata@zs9elektronik.pl.

```
print("Program sumujący podane liczby przez użytkownia.")
suma = 0
ile = int(input("Ile liczb chcesz zsumować? "))
licznik = 0
while licznik < ile:
    licznik += 1
    liczba = float(input(f"Podaj liczbę {licznik}: "))
    suma += liczba
print(f"Suma podanych liczb wynosi: ", suma)
print('Koniec programu.')
#
#
```

Wykonaj zadanie:

Napisz program z pętlą **while** znajdujący największy wspólny dzielnik (NWD) dwóch liczb naturalnych (algorytm Euklidesa). Pracę zapisz w pliku pod nazwą **$nazwisko_$klasa_$gr_euklides.py** i prześlij do nauczyciela w postaci załącznika na adres greszata@zs9elektronik.pl.

```
#
print('Program znajdujący największy wspólny dzielnik (NWD, algorytm Euklidesa ).')
a = int(input('Podaj pierwszą liczbę maturalną: '))
b = int(input('Podaj drugą liczbę naturalną: '))
x, y = a, b
while a != b:
    if a > b:
        a = a - b
    else:
        b = b - a
print('NWD liczb: ', x, ' oraz ', y, ' jest: ',a)
#
#
```

Temat: Pętla programowa for.

Pętla for jest uniwersalnym iteratorem, instrukcje wewnątrz pętli powtarzane są przez uporządkowany licznik w dowolnej sekwencji lub innym obiekcie iterowanym (np. łańcuchu znaków, liście, krotce).
Ogólna postać pętli for:

```
for <cel> in <obiekt>:
    <instrukcje1>
else:                   #alternatywa opcjonalna
    <instrukcje2>       #instrukcje wykonane, jeżeli nie było break
```

Przykłady zastosowania pętli for:

```
for x in [1, 2, 3]:
   print(x)             #domyślnie po każdym wydruku następuje przejście do nowej linii
#
for x in [1, 2, 3]:
   print(x, end = ' ')  #zastąpienie domyślnego \n na spację ' '

#
for x in (1, 2, 3):
   print(x ** 2, end = ' ')
#
for x in '1,2,3':
   print(x * 2, end = ' ')
#
for i in range(6):
   print(i, 'pozdrowienia')
#
#
#instrukcja range
range(5, 10)
#   5, 6, 7, 8, 9
#
range(0, 10, 3)
#   0, 3, 6, 9
#
range(-10, -100, -30)
#  -10, -40, -70
#
#
tablica = []
for x in range(5):
    if x % 2 == 0:
        for y in range(5):
            if y % 2 == 0:
                tablica.append((x, y))    #dodanie pary liczb do listy
print(tablica)
#
#
for z in 'Koszalin':
```

```
    print(z)
#
#
for z in 'Koszalin':
    print(z.upper())
#
#words = ['cat', 'window', 'defenestrate']
for w in words:
    print(w, len(w))
#wynik działania powyższego programu:
#cat 3
#window 6
#defenestrate 12
#
#
a = ['Mary', 'had', 'a', 'little', 'lamb']
for i in range(len(a)):
    print(i, a[i])
#wynik działania powyższego programu:
#0 Mary
#1 had
#2 a
#3 little
#4 lamb
#
#
lista = [1, 2, 3, 4, 5]
suma = 0
for x in lista:
    suma += x
print(suma)
#
#
kwadraty = [x ** 2 for x in [1, 2, 3, 4, 5]]
print(kwadraty)
#
#
kwadraty = []
for x in [1, 2, 3, 4, 5]:
    kwadraty.append(x ** 2)         #dodanie wartości do listy kwadraty
print(kwadraty)
#
#
#Iteracja wykorzystująca wycinek listy (fragment):
gracze = ['jola', 'adam', 'marek', 'ala', 'ela']
print('Lista trzech pierwszych graczy naszej szkoły:')
for gracz in gracze[:3]:
    print('- ', gracz.title())
#
#
#Przechowywanie informacji o pizzy zamawianej przez klienta w liście
pizza = {
    'ciasto': 'grubym',
    'dodatki': ['pieczarki', 'podwójny ser']
    }
#Podsumowanie zamówienia
print('Zamowiłes pizze na ' + pizza['ciasto'] + ' cieście ' +
    'wraz z następującymi dodatkami: ')
#Lista dodatków do pizzy
for dodatek in pizza['dodatki']:
    print('\t' + dodatek)
print('\nTwoja pizza jest już gotowa.')
#
#
#Działanie instrukcji for z listą
zamowienie = ['pieczarki', 'pepperoni', 'boczek']
for dodatek in zamowienie:
    if dodatek == 'boczek':
```

```
        print('Przepraszamy, ale nie mamy boczku.')
    else:
        print('Dodaję do pizzy ' + dodatek + '.')
print('\nTwoja pizza jest już gotowa.')
#
#
#Działanie instrukcji for i if z dwiema listami
dostepne_dodatki = ['pieczarki', 'pepperoni', 'boczek', 'podwójny ser']
zamowienie = ['pieczarki', 'frytki', 'pepperoni']
for dodatek in zamowienie:
    if dodatek in dostepne_dodatki:
        print('Dodaję do pizzy ' + dodatek + '.')
    else:
        print('Przepraszamy, ale nie mamy dodatku ' + dodatek + '.')

print('\nTwoja pizza jest już gotowa.')
#
#
samochody = ['audi', 'bmw', 'fiat']
for auto in samochody:
    if auto == 'bmw':
        print(samochody.upper())
    else:
        print(samochody.title())
#
#
#Działanie instrukcji for ze słownikiem
ulubione_jezyki = {
    'janek': ['python', 'c'],
    'ania': ['c'],
    'tomek': ['pascal', 'c', 'java'],
    'edward': ['ruby', 'go']
    }
#do wyświetlenia wartości kluczy stosujemy podwójna pętle for
for imie, jezyki in ulubione_jezyki.items():
    print('\nUlubiene jezyki programowania ' + imie.title() +
    ' to:')
    for jezyk in jezyki:
        print('\t' + jezyk.title())
#
#
#Działanie instrukcji for ze słownikiem
users = {
    'tomgres': {
        'imie': 'tomek',
        'nazwisko': 'greszata',
        'miasto': 'koszalin'
        },
    'mcurie': {
        'imie': 'maria',
        'nazwisko': 'skłodowska-curie',
        'miasto': 'paryż'
        }
    }
for username, user_info in users.items():
    print('\nNazwa użytkownika: ' + username)
    full_name = user_info['imie'] + ' ' + user_info['nazwisko']
    miasto = user_info['miasto']
    print('\tImię i nazwisko: ' + full_name.title())
    print('\tMiejscowosc: ' + miasto.title())
#
#
#Tabliczka mnożenia
n = 10
print('Program wyświetla tabliczkę mnożenia dla liczb od 1 do 10.')
for wiersz in range(n):
    for kolumna in range(n):
        print((wiersz + 1) * (kolumna + 1), end = '\t')
```

```
    print()
#
#
#Tabliczka mnożenia
print('Program wyświetla tabliczkę mnożenia dla liczb od 1 do 10.')
for wiersz in range(1, 11, 1):
    for kolumna in range(1, 11):
        print(wiersz * kolumna, end = '\t')
    print()
#
#
tablica = [(1, 2), (3, 4), (5, 6)]
for (a, b) in tablica:
    print(a, b)
#
#
dniTygodnia = ['Poniedziałek', 'Wtorek', 'Środa', 'Czwartek', 'Piątek', 'Sobota',
'Niedziela']
for dzien in dniTygodnia[-2:]:
    print(dzien)
#
#
#Powitanie ponownie zalogowanych użytkowników w serwisie internetowym:
#Gdy lista 'users' jest pusta, wyświetli się stosowny komunikat.
#
users = ['adam', 'tomek', 'ewa', 'jola', 'admin']
#users = []
if users:
    for user in users:
        if user == 'admin':
            print('Witaj, ' + user.title() + '! Jakie zadania na dzisiaj?')
        else:
            print('Witaj, ' + user.title() + '! Dziękujemy, że ponownie się
zalogowałes.')
else:
    print('Brak użytkowników.')
#
#
#Program sprawdzający, czy dana nazwa uzytkownika jest zajęta:
#
users = ['adam', 'tomek', 'ewa', 'jola', 'admin']
new_users = ['zenek', 'mariola', 'adam']
for user in new_users:
    if user in users:
        print('Podana nazwa użytkownika: ' + user + ' jest zajęta.')
    else:
        print('Witamy nowego użytkownika, ' + user + '.')
pass
#
#
#Program sprawdzający, czy liczby z podanego zakresu są liczbami pierwszymi
for n in range(2, 10):
    for x in range(2, n):     #jak to ulepszyć?
        if n % x == 0:        #reszta z dzielenia
            print(n, 'equals', x, '*', n//x)
            break
    else:
        #loop fell through without finding a factor
        #pętla przeszła całość nie znajdując dzielnika
        print(n, 'is a prime number')     #jest liczbą pierwszą
#wynik działania powyższego programu:
#2 is a prime number
#3 is a prime number
#4 equals 2 * 2
#5 is a prime number
#6 equals 2 * 3
#7 is a prime number
#8 equals 2 * 4
```

```
#9 equals 3 * 3
#
#
```

Ogólny format zatrzymania pętli for:

```
for <cel> in <obiekt>:
    <instrukcje1>
    if <test2>: break         #wyjście z pętli
    if <test3>: continue      #przechodzi do początku pętli
else:
    <instrukcje2>
```

Przykłady zastosowania pętli for z instrukcjami braek i continue:

```
#
for i in range(100):
    if i==50:
        brake
    print(i)
#
#
#wyznaczanie liczb pierwszych z podanego zakresu
for y in range(20, 31):
    x = y // 2      #dla y > 0
    while x > 1:
        if y % x == 0:
            print(y, ' ma dzielnik', x)
            break
        x -= 1
    else:
        print(y, ' jest liczba pierwsza.')
print('Koniec programu.')
#
#
```

Wykonaj zadanie:

Napisz program z pętlą for obliczający sumę tylu liczb i takich, jakie poda użytkownik.

```
#
n = int(input("Ile liczb chcesz dodać? "))
suma = 0
for i in range(n):
    liczba = float(input(f"Wprowadź liczbę {i + 1}: "))
    suma += liczba
print(f"Suma wprowadzonych liczb wynosi: {suma}")
#
```

Wykonaj zadanie:

Napisz program z pętlą for znajdujący największy wspólny dzielnik (NWD) dwóch liczb naturalnych (algorytm Euklidesa). Pracę zapisz w pliku pod nazwą **$nazwisko_$klasa_$gr_euklides.py** i prześlij do nauczyciela w postaci załącznika na adres greszata@zs9elektronik.pl.

```
#
print('Program znajdujący największy wspólny dzielnik (NWD, algorytm Euklidesa).')
a = int(input('Podaj pierwszą liczbę maturalną: '))
b = int(input('Podaj drugą liczbę naturalną: '))
for x in range(min(a, b), 0, -1):
    if a % x == 0 and b % x == 0:
        break
print('NWD liczb: ', a, ' i ', b, ' jest: ',x)
print('Koniec programu.')
#
#
```

Temat: Zarządzanie plikami w Pythonie.

Obiekty plików są w Pythonie interfejsem do plików zapisanych na dysku komputera. Do zarządzania plikami używamy następujących funkcji:

```
plik = open('a:\\nazwa_pliku.txt', 'w')   #utworzenie pliku w trybie do zapisu
plik.write('Witaj!')                 #zapisanie danych do pliku (kursor pozostaje)
plik.write('Witaj!\n')                      #wymuszenie przejścia do nowej linii
print('Kurs Pythona.\n', file = plik)     #przekierowanie danych do pliku
plik.close()                                #zamknięcie pliku i zapisanie na dysku
plik.write('Koniec pliku.\n')               #nadpisanie pliku - 'w' (write)
plik.write('Koniec pliku.\n')               #dopisanie tekstu do pliku - 'a' (append)
plik.write('Koniec pliku.\n')               #zablokowany zapis do pliku - 'r' (read)
#
#
plik = open('C:\\programy\\ plik.txt', 'w')     #\\ - wymusza poprawny odczyt ścieżki
plik = open(r'C:\programy\plik.txt', 'w')       #r – zapobiega odczytom \t, \n
plik = open('C:/programy/plik.txt', 'w')        #można także stosować znaki slash (/)
#
#
open('c:\\programy\\plik.txt', 'w').write('Pierwsza linijka\n')  #nadpisanie pliku
open('c:\\programy\\plik.txt', 'a').write('Druga linijka\n')     #dodanie tekstu
open('c:\\programy\\plik.txt', 'a').write('Trzecia linijka')     #kolejne dodanie
print(open('c:\\programy\\plik.txt').read())
open('c:\\programy\\plik.txt').close()
#
#
plik = open('nazwa_pliku.txt', 'r')       #otworzenie pliku w trybie tylko do odczytu
print(plik.read())
plik.close()
#
print(open('nazwa_pliku.txt').read())
#plik.close()
#
#
plik = open('nazwa_pliku.txt')            #domyślnie tylko do odczytu (tryb 'r')
tekst = plik.read()
print(tekst)
plik.close()
#
plik = open('nazwa_pliku.txt')
while True:
    znak = plik.read(1)                   #wczytanie znak po znaku
    if not znak:
        break
    elif znak == ' ':
        znak = '\n'
    print(znak, end = '')
plik.close()
#
for znak in open('nazwa_pliku.txt'):
    print(znak)
#
plik = open('nazwa_pliku.txt')
print(plik.readline())                    #wydruk pierwszej linii
print(plik.readline())                    #wydruk drugiej linii
print(plik.readline())                    #wydruk kolejnej linii
plik.close()
#
#
plik = open('nazwa_pliku.txt')
while True:
    wiersz = plik.readline(1)                   #wczytanie wiersz po wierszu
    if not wiersz: break
    print(wiersz, end = ' ')
plik.close()
#
#
for wiersz in open('nazwa_pliku.txt'):
    print(wiersz, end = ' ')
plik.close()
```

```
#
#
```

Inna metoda wczytania pliku polega na użyciu polecenia (słowa kluczowego) **with**, które powoduje automatyczne zamknięcie pliku, gdy dostęp do niego nie będzie potrzebny. Do pracy z plikami zaleca się stosowanie metody with. Przykłady zastosowania:

```
with open('nazwa_pliku.txt') as odczyt:
    zawartoscPliku = odczyt.read()
print(zawartoscPliku)
#
#
with open('katalog/nazwa_pliku.txt', encoding = 'utf-8') as odczyt:
    zawartoscPliku = odczyt.read()
print(zawartoscPliku)
#
#
def osoba(nr, nazwisko, imie):
    return f"{nr} {nazwisko} {imie}"

    for x in range(1, 4, 1):
        print("\nPodaj imię i nazwisko ucznia ", x,":"
        n = input("nazwisko: ")
        i = input("imie: ")
        with open('osoby.txt', '+') as o:
            o.write(x, n, i, '\n')
        k = input("Wpisz 'k', aby zakończyć wprowadzanie.")
        if k == 'k':
            break

with open('osoby.txt', 'r') as o:
    print("Aktualna lista uczniów:")
    print(o.read())
#
#
sciezkaDoPliku = 'plik.txt'
```

Temat: Funkcje w Pythonie.

Funkcje służą do wielokrotnego wykorzystania kodu bez konieczności ponownego zapisania kodu.

Ogólna postać funkcji:

```
def nazwa_funkcji(arg1, arg2, ...argn):   #argumenty są opcjonalne
    instrukcje           #wcięcia decydują o przynależności instrukcji do bloku
    return wartość       #opcjonalnie, odsyła wynik do programu,
                         #gdy brak return, funkcja odsyła wartość None (ignorowane)
nazwa_funkcji()          #wywołanie funkcji
```

Nazwy lokalne używane w funkcjach nadpisują nazwy globalne, np.:

```
x = 11                   #zmienna globalna, x dla całego pliku
def funkcja():
    x = 22               #zmienna lokalna, x działa tylko w funkcji
#
```

Przykłady prostych funkcji:

```
def greet_user(username):
    """Funkcja Wyswietla proste powitanie."""
    print('Witaj! ' + username.title())
#Wywołanie funkcji z podaniem argumentu 'tomek'.
greet_user('tomek')
#
#
#Definicja funckji z nazwą parametrow 'imie' i 'nazwisko'.
def greet_user(imie, nazwisko):
    """Funkcja Wyswietla proste powitanie z iminiem i nazwiskiem."""
    print('Witaj, ' + imie.title() + ' ' + nazwisko.title() + '!')
```

```
#Wywołanie funkcji z podaniem dwóch argumentów.
greet_user('tomek', 'greszata')
#Inne sposoby wywołania funkcji ze zmianą pozycji argumentów.
#Kolejność podania argumentów jest istotna.
greet_user('greszata', 'tomek')
#Wywołanie funkcji z precyzyjnym podaniem argumentów
#Kolejność podania argumentów nie jest istotna.
greet_user(imie='tomek', nazwisko = 'greszata')
greet_user(nazwisko = 'greszata', imie='tomek')
#
#
#Definicja funkcji zwracającej wartość.
def get_formatted_name(imie, nazwisko):
    """Funkcja zwraca sformatowany tekst z imieniem i nazwiskiem."""
    pelna_nazwa = imie + ' ' + nazwisko
    return pelna_nazwa.title()
#Wywołanie funkcji przez zmienną
osoba = get_formatted_name('tomek', 'greszata')
print(osoba)
#
#
#Definicja funkcji zwracającej wartość. Argument drugie_imie ma wartość domyślną.
def get_formatted_name(pierwsze_imie, nazwisko, drugie_imie =''):
    """Funkcja zwraca sformatowany tekst z iminiem i nazwiskiem."""
    if drugie_imie:
        pelna_nazwa = pierwsze_imie + ' ' + drugie_imie + ' ' + nazwisko
    else:
        pelna_nazwa = pierwsze_imie + ' ' + nazwisko
    return pelna_nazwa.title()
#Wywołanie funkcji
osoba = get_formatted_name('tomasz', 'greszata', 'ireneusz')
print(osoba)
#Wywołanie funkcji dla osoby z jednym imieniem.
osoba = get_formatted_name('janusz', 'kowalski')
print(osoba)
#
#
```

Przykłady zastosowania funkcji:

```
def osoba(typ = "pracownik", imie, nazwisko):  #z wartością domyślną typ
    print(f'{typ.title()} - Imie: {imie}, nazwisko: {nazwisko}')
osoba("pracownik", "Jan", "Kowalski")     #przykładowe wywołanie funkcji
osoba(typ = "pracownik", imie = "Jan", nazwisko = "Kowalski") #klucz-wartość
#
#
def wspolne(arg1, arg2):
    tablica = []
    for x in arg1:
        if x in arg2:
            tablica.append(x)
    return tablica
print(wspolne('Tomek', 'Tymon'))
x = wspolne([1, 2, 3, 4, 5], (1, 4))
print(x)
#
#
#Program obliczający sumę liczb podanych przez użytkownika
def suma_liczb():
    # Prośba o podanie liczby
    n = int(input("Ile liczb chcesz dodać? "))
    suma = 0
    for i in range(n):
        liczba = float(input(f"Wprowadź liczbę {i + 1}: "))
        suma += liczba
    print(f"Suma wprowadzonych liczb wynosi: {suma}")
suma_liczb()            #wywołanie funkcji bez argumentu
#
#
```

```
def kwadraty(n):
    tablica = []
    for i in range(n):
        tablica.append(i ** 2)
    return tablica
for x in kwadraty(5):   #wielokrotne wywołanie funkcji bez z argumentem
    print(x, end=' ')
#
#
#Program obliczający sumę liczb podanych w postaci listy
def suma_liczb(lista):
    if not lista:
        return 0
    else:
        return (lista[0] + suma_liczb(lista[1:]))
print(suma_liczb([1, 2, 3, 4, 5]))        #wywołanie funkcji z argumentem
#
#
def silnia(i):          #definicja funkcji silnia dla liczb naturalnych
    if i == 0:
        return 1        #wartość zwracana przez funkcję (gdy brak, to null)
    else:
        return i * silnia(i - 1)
print(silnia(6))        #wywołanie funkcji z argumentem
#
#
def silnia(x):
    if x <= 1:
        return 1
    return x * silnia(x - 1)
print(silnia(6))
#
#
def NWD(a, b):
    while a != b:
        if a > b:
            a = a - b
        else:
            b = b - a
    return a
#
#
print('Program znajdujący największy wspólny dzielnik (NWD, algorytm Euklidesa ).')
def NWD(a, b):
    if a != b:
        if a > b:
            return NWD(a - b, b)
        else:
            return NWD(a, b - a)
    return a
print(NWD(27,9))
#
#
def fib(n):       #definicja funkcji dla ciągu Fibonacciego dla liczb naturalnych
    if n == 0:
        return 0
    elif n == 1:
        return 1
    else:
        return fib(n - 1) + fib(n - 2)
#
#
#losowanie liczb lotto
def lotto(a):
    import random
    for i in range(a):
        liczba = random.randint(1, 49)
        print(liczba, end = " ")
```

```
lotto(6)
#
#
#losowanie liczb lotto i sortowanie roznąco
def lotto(a):
    import random
    los = []
    for i in range(a):
        liczba = random.randint(1, 49)
        los.append(liczba)
    return sorter(los)
print(lotto(6))
#
#
#sortowanie kubełkowe
def sort_kubelk(liczby):
    # znajdowanie wartości minimalnej i maksymalnej
    n = len(liczby)
    min_elem = liczby[0]
    max_elem = liczby[0]
    for i in range(1, n):
        if liczby[i] < min_elem:
            min_elem = liczby[i]
        if liczby[i] > max_elem:
            max_elem = liczby[i]
    # przygotowanie listy zliczającej
    k = max_elem - min_elem + 1
    kubelek = [0] * k
    # zliczanie wartości
    for elem in liczby:
        kubelek[elem - min_elem] += 1
    # porządkowanie
    posortowane_elem = []
    for i in range(k):
##    for i in range(k - 1, -1, -1):  #odwrócenie sortowania
        while kubelek[i] > 0:
            posortowane_elem.append(i + min_elem)
            kubelek[i] -= 1
    return posortowane_elem

liczby = [0, 2, 6, 4, -4, 3, 1, 2, 5, -5]
print(sort_kubelk(liczby))
#
#
def osoba(imie, nazwisko):
    return f"{imie} {nazwisko}"
    while True:
        print("\nPodaj imię i nazwisko:"
        print("Wpisz 'k', aby zakończyć wprowadzanie.")
        i = input("Imię: ")
        if i == 'k':
            break
        n = input("Nazwisko: ")
        if n == 'k':
            break
o = osoba(i, n)
print(f"\nWitaj, {o}!")
#
#
```

Temat: Moduły w Pythonie.

Moduły to jednostki najwyższego poziomu organizacji programu. Moduły zawierają kod programu i dane, które mogą być użyte w dowolnym miejscu programu. Zazwyczaj moduły to pliki programów Pythona. Moduły przetwarzane są za pomocą następujących instrukcji:

```
import nazwa_modulu                        #pobranie całego modułu
from nazwa_modułu import nazwa_funkcji     #pobranie określonych nazw z modułu
```

Przykład wywołania pomocy na temat modułu lub funkcji:

```
help(nazwa_modulu)
help(nazwa_modulu.nazwa_funkcji)
modul.klasa.metoda.__doc__          #pomoc dla metody (funkcji)
print(modul.__doc__)                #wcześniej należy importować moduł
print(dir(modul))                   #wyświetlenie dostępnych klas (funkcji) w module
```

Przykłady zastosowania modułów:

```
from decimal import Decimal as D
a = D("0.1")
b = D("0.2")
c = a + b
if c == D("0.3"):
    print("OK - nasz wynik to 0.3 - oczekiwany!")
else:
    print(f"Nasz wynik jest dziwny: {c}")
#
#
import funkcje          #moduł zdefiniowany przez użytkownika w katalogu z programem
print(f"Pole kwadratu = {funkcje.poleKwadratu(3)}")
#
#
from funkcje import poleProstokata, poleKwadratu
print(poleProstokata(3, 5))
print(poleKwadratu(3))
#
#
from funkcje imort dodaj, poleKwadratu
poleKwadratu(4)
#
import funkcje as f
f.dodaj(3, 4)
#
#
import builtins         #import wbudowanego modułu
print(dir(builtins))    #lista nazw zdefiniowanych (funkcji)
#
#
import sys
print(sys.path)
sys.path.append('x:\\katalog_modulow')    #dodanie ścieżki do przeszukiwania
import funkcje
#
#
#sprawdzenie czasu wykonania funkcji
import datetime                     ##print(dir(datetime))
print(datetime.datetime.now())
start = datetime.datetime.now()
funkcja_do_sprawdzenia()
print('\nCzas: ', datetime.datetime.now() - start)
#
#
```

Przykłady zastosowania modułów. Załóżmy, że mamy następujący plik **funkcje.py**:

```
def tabliczkaMnozenia():
    print('Program tabliczka mnożenia.')
    i = 1
    while i <= 10:
        j = 1
        while j <= 10:
            print(i * j, end = '\t')
            j += 1
        print()
        i += 1
#
def silnia(x):
```

```
    if x <= 1:
        return 1
    else:
        return x * silnia(x - 1)
#
def napis(nap):
    while nap:
        print(nap, end = ' ')
        nap = nap[:-1]
#
def czytanie():
    plik = open('plik.txt')
    while True:
        znak = plik.read(1)
        if not znak:
            break
        elif znak == ' ':
            znak = '\n'
        print(znak, end = '')
    print('\n')
    plik.close()
#
def liczby_pierwsze(y):
    if y <= 0:
        print(y, ' nie jest liczba naturalna.')
    elif y == 1:
        print(y, ' nie jest liczba pierwsza.')
    else:
        x = y //2
        while x > 1:
            if y % x == 0:
                print(y, ' ma dzielnik rowny ', x)
                break
            x -= 1
        else:
            print(y, ' jest liczba pierwsza.')
    return 0
#
#
```

Przykłady odwołania się do funkcji z modułu z pliku **funkcje.py**:

```
import funkcje
funckje.tabliczkaMnozenia()
print(funkcje.silnia(5))
#
import funkcje as f
f.tabliczkaMnozenia()
#
from funkcje import *
tabliczkaMnozenia()
napis('Haslo.')
#
from funkcje import tabliczkaMnozenia()
tabliczkaMnozenia()
#
from funkcje import czytanie, silnia
print(silnia(5))
czytanie()
#
from funkcje import tabliczkaMnozenia as tm
tm()

from funkcje import liczby_pierwsze as lp
lp(7)
#
#
```

Wykonaj zadanie:

Napisz program wywołujący zdefiniowane trzy funkcje w pliku **funkcje.py** dowolnymi trzema różnymi metodami. Pracę zapisz w pliku pod nazwą **$nazwisko_$klasa_$gr_moduly.py** i prześlij do nauczyciela w postaci załącznika na adres greszata@zs9elektronik.pl.

Wykonaj zadanie:

Napisz program zgodny z hipotezą Goldbacha (każda liczba parzysta większa od dwóch jest sumą dwóch liczba pierwszych) dla liczby (zakresu liczb) podanej przez użytkownika.

Temat: Okna w Pythonie (GUI).

Moduł `tkinter` w Pythonie jest standardowym modułem do tworzenia GUI (graficznego interfejsu użytkownika).
Jak uruchomić programy Pythona:

- Upewnij się, że masz zainstalowany Python.
- Skopiuj kod do pliku z rozszerzeniem .py.
- Uruchom plik w środowisku, które wspiera GUI, np. na komputerze lokalnym.

Najprostsze okno w GUI:

```
from tkinter import *
#utworzenie okna
okno = Tk()
#kontrolka z napisem
etykieta = Label(okno, text='Lubię pai!')
#umieści etykietę w oknie z dopasowaniem automatycznym
etykieta.pack()
#wywołanie okna z oczekiwaniem na jego zamkniecie
okno.mainloop()
#
#
```

Drobna modyfikacja najprostszego okna w GUI:

```
from tkinter import *
okno = Tk()
#font - wybor formatu czcionki - kroj, wielkosc, pochylenie, kolor, kolor tla
etykieta = Label(okno, text="Hej!", font=("Arial",24,"italic"), foreground="yellow",
background="blue")
etykieta.pack(expand=YES, fill=BOTH)
okno.mainloop()
#
#
```

Przykład zastosowania okienek w Pythonie:

```
import tkinter as tk
# Tworzenie głównego okna
root = tk.Tk()
# Ustawienie tytułu okna
root.title("Moja pierwsza aplikacja w Tkinter")
# Ustawienie rozmiaru okna
root.geometry("400x300")  # szerokość x wysokość
# Ustawienie minimalnego i maksymalnego rozmiaru okna
root.minsize(200, 150)
root.maxsize(600, 400)
# Dodanie etykiety
label = tk.Label(root, text="Witaj w Tkinter!", font=("Helvetica", 16))
label.pack(pady=20)
# Dodanie przycisku
def on_button_click():
    label.config(text="Przycisk został naciśnięty!")
button = tk.Button(root, text="Naciśnij mnie", command=on_button_click)
button.pack(pady=10)
# Dodanie pola tekstowego
entry = tk.Entry(root)
entry.pack(pady=10)
```

```
# Uruchomienie głównej pętli aplikacji
root.mainloop()
#
#
```

Wyjaśnienie powyższego kodu:

- Importowanie Tkinter: Zaczynamy od zaimportowania modułu tkinter.
- Tworzenie głównego okna: `root = tk.Tk()` tworzy główne okno aplikacji.
- Konfiguracja okna:
  - Tytuł ustalamy za pomocą `root.title()`.
  - Rozmiar okna ustawiamy z użyciem `root.geometry()`.
  - Minimalny rozmiar okna możemy ustawić faktycznymi metodami `root.minsize()`.
- Dodawanie widgetów:
  - `Label` - wyświetla tekst.
  - `Button` - dodaje przycisk. Używamy metody `command`, aby określić, co ma się stać po kliknięciu przycisku.
  - `Entry` - pola tekstowe pozwalają użytkownikom wprowadzać tekst.
- Pętla główna: `root.mainloop()` uruchamia aplikację i czeka na zdarzenia użytkownika (np. kliknięcia).

Dodatkowe opcje konfiguracyjne:

- Kolory i czcionki: Możesz zmieniać kolory tła, czcionki itp., używając opcji takich jak bg, fg, font.
  `label = tk.Label(root, text="Witaj w Tkinter!", bg="yellow", fg="blue", font=("Helvetica", 16))`
- Układ: Możesz używać różnych menedżerów układu jak `pack()`, `grid()` i `place()`, aby kontrolować rozmieszczenie widgetów w oknie.
- Obsługa zdarzeń: Możesz dodawać własne funkcje do obsługi różnych zdarzeń, takich jak naciśnięcie klawisza, zwolnienie myszki itp.

W bibliotece Tkinter mamy kilka sposobów na regulowanie pozycji i rozmiarów obiektów w oknie. Wybór metody zależy od tego, jak bardzo precyzyjnie chcesz kontrolować pozycję i rozmiar swoich elementów. `pack()` i `grid()` są bardziej intuicyjne dla układów, podczas gdy `place()` daje pełną kontrolę, ale wymaga więcej pracy, by dostosować do zmian w rozmiarze okna.

Przykłady zarządzania geometrią w okienkach:

```
#pack() ustawia obiekty w kolejności, w jakiej zostały dodane,
#domyślnie od góry do dołu lub od lewej do prawej
#side, fill, expand, padx, pady - parametry do kontroli pozycji i rozmiaru
import tkinter as tk
root = tk.Tk()
label1 = tk.Label(root, text="Label 1")
label1.pack(side=tk.TOP, pady=10)
label2 = tk.Label(root, text="Label 2")
label2.pack(side=tk.TOP, pady=10)
root.mainloop()
#
#
# grid() pozwala na ułożenie obiektów w układzie siatki
#można określić wiersz (row) i kolumnę (column), w których ma znajdować się obiekt
import tkinter as tk
root = tk.Tk()
label1 = tk.Label(root, text="Label 1")
label1.grid(row=0, column=0, padx=10, pady=10)
label2 = tk.Label(root, text="Label 2")
label2.grid(row=0, column=1, padx=10, pady=10)
root.mainloop()
#
#
# place() daje dużą swobodę w określaniu dokładnej pozycji i rozmiaru obiektów
root.mainloop()
import tkinter as tk
```

```
root = tk.Tk()
label1 = tk.Label(root, text="Label 1")
label1.place(x=20, y=30, width=100, height=30)
label2 = tk.Label(root, text="Label 2")
label2.place(x=90, y=30, width=100, height=30)
label3 = tk.Label(root, text="Label 3")
label3.place(x=20, y=100, width=100, height=30)
label4 = tk.Label(root, text="Label 4")
label4.place(x=90, y=100, width=100, height=30)
root.mainloop()
#
```

Można regulować rozmiar widgetów, przeważnie z użyciem opcji takich jak `width` i `height` podczas tworzenia widgetu. Jednak wiele widgetów automatycznie dostosowuje swój rozmiar do treści. Można również ustawić opcje takie jak `pack_propagate` lub `grid_propagate` na `False`, aby zapobiec automatycznej zmianie rozmiaru kontenera.

W Tkinter metody rozmieszczania widgetów (`pack, grid, place`), nie powinny być używane w tym samym kontenerze (np. w tym samym oknie lub ramce). Można jednak stosować te metody w różnych kontenerach (ramkach). Jeśli różne metody rozmieszczania widgetów zostaną użyte w tym samym kontenerze, Tkinter zgłosi błąd.

Przykład, jak poprawnie łączyć metody (`pack, grid, place`):

```
import tkinter as tk
#polaczenie trzech metod pozycjonowania obiektow w oknie graficznym
root = tk.Tk()
#dodany rozmiar okna dla metody place
root.geometry("400x200")

### Tworzymy ramkę i używamy metody pack
frame1 = tk.Frame(root)
frame1.pack()
label1 = tk.Label(frame1, text="To jest label w frame1 używając pack")
label1.pack()
button1 = tk.Button(frame1, text="Przycisk 1")
button1.pack()

# Tworzymy drugą ramkę i używamy metody grid
frame2 = tk.Frame(root)
frame2.pack()
label2 = tk.Label(frame2, text="To jest label w frame2 używając grid")
label2.grid(row=0, column=0)
button2 = tk.Button(frame2, text="Przycisk 2")
button2.grid(row=1, column=0)

# Tworzymy trzecią ramkę i używamy metody place
##frame3 = tk.Frame(root)
##frame3.pack()
##label3 = tk.Label(frame3, text="To jest label w frame3 używając place")
##label3.place(x=20, y=120)
label3 = tk.Label(root, text="To jest label w frame3 używając place")
label3.place(x=100, y=100, width=200, height=30)
button3 = tk.Button(root, text="Przycisk 3")
button3.place(x=170, y=130)

root.mainloop()
```

Przykład zastosowania okienek w Pythonie:

```
##Dodawanie dwoch liczb
import tkinter
from tkinter import *
from tkinter import ttk
## Funkcja odpowiedzialna za wygląd okna
def wyswietl():
    okno = Tk()
```

```
        labelkaPierwszaLiczba = Label(okno, text='Pierwsza liczba')
        labelkaPierwszaLiczba.grid(padx=5, pady=5, row=0,column=0)

        polePierwszaLiczba = Entry(okno, width=10)
        polePierwszaLiczba.grid(sticky=E, padx=5,pady=5,row=0,column=1)

        labelkaDrugaLiczba = Label(okno, text='Druga liczba')
        labelkaDrugaLiczba.grid(sticky=E, padx=5, pady=5, row=1, column = 0)

        poleDrugaLiczba = Entry(okno, width=10)
        poleDrugaLiczba.grid(padx=5, pady=5, row=1, column=1)

        def funkcjaDodaj():
            pierwszaLiczba = 0
            try:
                pierwszaLiczba = float(polePierwszaLiczba.get())
            except ValueError:
                labelkaWynik.config(text="złe dane")

            drugaLiczba = 0
            try:
                drugaLiczba = float(poleDrugaLiczba.get())
            except ValueError:
                labelkaWynik.config(text="złe dane")

            finally:
                wynik = pierwszaLiczba + drugaLiczba
            labelkaWynik.config(text=str(wynik))

        przyciskDodaj = Button(okno, text ='Dodaj liczby', command=funkcjaDodaj)
        przyciskDodaj.grid(sticky=E + W, padx=5, pady=5,row=3,column=0,columnspan=2)

        labelkaWynik = Label(okno, text='Wynik')
        labelkaWynik.grid(sticky=E + W, padx=5,pady=5,row=4,column=0,columnspan=2)
        okno.mainloop()

    wyswietl()
    #
```

Przykład zastosowania okienek w Pythonie:

```
    ## Projekt kalkulatora
    ##
    from tkinter import *
    from tkinter import ttk
    okno = Tk()
    okno.geometry("400x300")
    okno.title("Kalkulator")

    dane1 = Label(okno, text = "Wpisz pierwszą liczbę:")
    dane1.grid(row=0, column=0)
    poleDane1 = Entry(okno)
    poleDane1.grid(row=0, column=1)

    dane2 = Label(okno, text = "Wpisz drugą liczbę:")
    dane2.grid(row=1, column=0)
    poleDane2 = Entry(okno)
    poleDane2.grid(row=1, column=1)

    ## pole wynik
    wynik = Label(okno, text = "Wynik działania:")
    wynik.grid(row=3, column=0)
    poleWynik = Entry(okno)
    poleWynik.grid(row=3, column=1)

    def przyciskDodawanie():
        # poleWynik.config(text = int(poleDane1.get()) + int(poleDane2.get()))
        poleWynik.delete(0, len(poleWynik.get()))
        poleWynik.insert(0,str(int(poleDane1.get()) + int(poleDane2.get())))
```

```
def przyciskOdejmowanie():
    poleWynik.delete(0, len(poleWynik.get()))
    poleWynik.insert(0,str(int(poleDane1.get()) - int(poleDane2.get())))

def przyciskMnozenie():
    poleWynik.delete(0, len(poleWynik.get()))
    poleWynik.insert(0,str(int(poleDane1.get()) * int(poleDane2.get())))

def przyciskDzielenie():
    poleWynik.delete(0,len(poleWynik.get()))
    poleWynik.insert(0,'%.2f' % (int(poleDane1.get()) / int(poleDane2.get())))

plus = Button(okno, text= " + ", command = przyciskDodawanie, width=5, height=2)
plus.grid(row=0, column=2)

minus = Button(okno, text= " - ", command = przyciskOdejmowanie, width=5, height=2)
minus.grid(row=1, column=2)

mnoz = Button(okno, text= " * ", command = przyciskMnozenie, width=5, height=2)
mnoz.grid(row=2, column=2)

dziel = Button(okno, text= " / ", command = przyciskDzielenie, width=5, height=2)
dziel.grid(row=3, column=2)

okno.mainloop()
#
```

Przykład zastosowania okienek w Pythonie (obsługa pól wyboru):

```
import tkinter
from tkinter import messagebox
import tkinter
import tkinter.messagebox
class MyGUI:
    def __init__(self):
##      Utworzenie okna głównego
        self.main_window = tkinter.Tk()

##      Utworzenie dwóch kontenerów. Jeden dla widżetów Radiobutton,
##      natomiastdrugi dla zwykłych widżetów Button.
        self.top_frame = tkinter.Frame(self.main_window)
        self.bottom_frame = tkinter.Frame(self.main_window)

##      Utworzenie obiektu IntVar przeznaczonego
##      do użycia wraz z widżetami Radiobutton.
        self.radio_var = tkinter.IntVar()

##      Przypisanie obiektowi IntVar wartości 1.
        self.radio_var.set(1)

##      Utworzenie widżetów Rodiobutton w kontenerze top_frame.
        self.rb1 = tkinter.Radiobutton(self.top_frame,
                                       text = 'Opcja 1',
                                       variable=self.radio_var,
                                       value=1)
        self.rb2 = tkinter.Radiobutton(self.top_frame,
                                       text='Opcja 2',
                                       # połączenie przycisku ze zmienną,
                                       variable=self.radio_var,
                                       value=2)
        self.rb3 = tkinter.Radiobutton(self.top_frame,
                                       text='Opcja 3',
                                       variable=self.radio_var,
                                       value=3)

##      Wywołanie metody pack() widżetów Radiobutton.
        self.rb1.pack()
        self.rb2.pack()
```

```
        self.rb3.pack()

##      Utworzenie przycisków OK i Zakończ.
        self.ok_button = tkinter.Button(self.bottom_frame,
                                        text='OK',
                                        command=self.show_choice)
        self.quit_button = tkinter.Button(self.bottom_frame,
                                        text='Zakończ',
                                        command=self.main_window.destroy)

##      Wywołanie metody pack() widżetów Button.
        self.ok_button.pack(side='left')
        self.quit_button.pack(side='left')

##      Wywołanie metody pack() kontenerów.
        self.top_frame.pack()
        self.bottom_frame.pack()

##      Wywołanie metody mainloop() modułu tkinter.
        tkinter.mainloop()

##     Metoda show_choice() jest funkcją wywołania zwrotnego dla przycisku OK.
    def show_choice(self):
        tkinter.messagebox.showinfo('Wybór',      'Wybrałeś      opcję      '      +
str(self.radio_var.get()))

## Utworrzenie egzemplarza klasy MyGUI.
my_gui = MyGUI()
```

Przykład zastosowania okienek w Pythonie (wyświetlenie zawartości pliku):

```
##Wyświetlenie zawartości pliku tekstowego w oknie graficznym. Aby wyświetlić plik
tekstowy w oknie aplikacji stworzonej z użyciem biblioteki Tkinter w Pythonie, można
użyć widżetu "Text". Poniżej znajduje się prosty przykład, jak to zrobić:
##
import tkinter as tk
from tkinter import scrolledtext
from tkinter import filedialog

def open_file():
    file_path = filedialog.askopenfilename(filetypes=[("Text files", "*.txt")])
##    file_path = 'a:\\python\\plik.txt'
    if file_path:
        with open(file_path, 'r', encoding='utf-8') as file:
            content = file.read()
            text_area.delete(1.0, tk.END) # Wyczyść poprzednią zawartość
            text_area.insert(tk.END, content) # Wstaw nową zawartość

# Utwórz główne okno
root = tk.Tk()
root.title("Wyświetlanie pliku tekstowego")

# Utwórz przycisk do otwierania pliku
open_button = tk.Button(root, text="Otwórz plik tekstowy", command=open_file)
open_button.pack(pady=10)

# Utwórz obszar tekstowy do wyświetlania zawartości pliku, 20 wierszy, 60 znaków,
text_area = scrolledtext.ScrolledText(root, wrap=tk.WORD, width=60, height=20)
text_area.pack(padx=10, pady=10)

# Rozpocznij główną pętlę programu
root.mainloop()
#
#
```

Opis powyższego kodu:

1. Importujemy potrzebne moduły z biblioteki `tkinter` oraz `filedialog`.

2. Funkcja `open_file()` otwiera okno dialogowe, pozwalając użytkownikowi na wybór pliku tekstowego. Po wybraniu pliku, jego zawartość jest odczytywana i wyświetlana w obszarze tekstowym.

3. Tworzymy główne okno aplikacji oraz przycisk, który uruchamia funkcję `open_file()`.
4. `ScrolledText` to rozszerzenie widżetu `Text`, które automatycznie dodaje pasek przewijania dla długich tekstów.
5. Całość kończymy wywołaniem `root.mainloop()`, które uruchamia pętlę wydarzeń aplikacji.

Dzięki temu prostemu programowi użytkownik może otworzyć i wyświetlić zawartość pliku tekstowego w przyjaznym interfejsie graficznym.

Przykład zastosowania okienek w Pythonie:

Program rozwiązujący równanie kwadratowe. Aby obliczyć pierwiastki funkcji kwadratowej w Tkinter, należy najpierw zdefiniować funkcję kwadratową w postaci ogólnej:

[ax^2 + bx + c = 0]

Pierwiastki tej funkcji można obliczyć przy użyciu wzoru kwadratowego:

[x_{1,2} = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}]

Możesz utworzyć prosty interfejs graficzny w Tkinter, aby wprowadzić wartości (a), (b) i (c) oraz wyświetlić wyniki. Oto przykładowa implementacja:

```
##Kod programu obliczającego pierwiastki funkcji kwadratowej:
import tkinter as tk
import math
def oblicz_pierwiastki():
    try:
        a = float(entry_a.get())
        b = float(entry_b.get())
        c = float(entry_c.get())
        delta = b**2 - 4*a*c
        if delta < 0:
            wynik.set("Brak pierwiastków rzeczywistych")
        elif delta == 0:
            x = -b / (2 * a)
            wynik.set(f"Pojedynczy pierwiastek: x = {x}")
        else:
            x1 = (-b + math.sqrt(delta)) / (2 * a)
            x2 = (-b - math.sqrt(delta)) / (2 * a)
            wynik.set(f"Pierwiastki: x1 = {x1:.2f}, x2 = {x2:.2f}")
    except ValueError:
        wynik.set("Proszę podać poprawne liczby")
# Tworzenie okna
root = tk.Tk()
root.title("Obliczanie pierwiastków funkcji kwadratowej")
# Tworzenie widżetów
tk.Label(root, text="a:").grid(row=0, column=0)
tk.Label(root, text="b:").grid(row=1, column=0)
tk.Label(root, text="c:").grid(row=2, column=0)
entry_a = tk.Entry(root)
entry_b = tk.Entry(root)
entry_c = tk.Entry(root)
entry_a.grid(row=0, column=1)
entry_b.grid(row=1, column=1)
entry_c.grid(row=2, column=1)
btn_oblicz = tk.Button(root, text="Oblicz pierwiastki", command=oblicz_pierwiastki)
btn_oblicz.grid(row=3, columnspan=2)
wynik = tk.StringVar()
tk.Label(root, textvariable=wynik).grid(row=4, columnspan=2)
# Uruchomienie pętli głównej
root.mainloop()
#
#
```

Opis powyższego kodu:

- Używamy biblioteki tkinter do stworzenia prostego GUI.
- Użytkownik wprowadza wartości dla (a), (b) i (c) w odpowiednich polach tekstowych.
- Po naciśnięciu przycisku "Oblicz pierwiastki" program oblicza deltę i na jej podstawie wylicza pierwiastki.
- Wynik jest wyświetlany w etykiecie na dole okna.

Wykonaj zadanie:

Napisz program w języku Python wykonujący następujące działania:

- utworzy plik tekstowy osoba.txt z Twoim imieniem i nazwiskiem (jeżeli w zadanym problemie występuje plik tekstowy, to Twoje imię i nazwisko wyświetl w okienku graficznym),
- wywoła okno graficzne z informacjami na temat zadania realizowanego przez program,
- wyświetli w oknie graficznym zawartość pliku osoba.txt,
- program główny wykorzysta zdefiniowaną w odrębnym pliku funkcję (najlepiej kilka),
- program będzie realizował jedno zadanie z poniższej listy:
    1. kalkulator z przyciskami cyfr dziesiętnych do wprowadzania liczb do działań matematycznych, Jacek
    2. kalkulator prosty działający w trybie okienkowym sterowany przyciskami +, -, *, /, oraz sprawdzający warunek zakazu dzielenia przez zero,
    3. podziel kolejne pary liczb podawanych przez użytkownika do momentu wystąpienia jako dzielnika wartości 0, liczby i wynik dzielenie zapisz do pliku wyniki.txt,
    4. odczytaj listę liczb zapisanych w pliku tekstowym dane.txt i wyznaczy z niej wartość najmniejszą i największą, a następnie dopisze je w kolejnych wierszach pliku źródłowego,
    5. odczytaj listę liczb zapisanych w pliku tekstowym dane.txt i posortuje je od największej do najmniejszej, a następnie zapisz tę posortowaną listę w pliku wynik.txt, wynik wyświetl w oknie graficznym,
    6. pobierz od użytkownika liczby do momentu wpisania znaku 'k' lub 'K', następnie posortuje je od największej do najmniejszej, a potem zapisz tę posortowaną listę w pliku wynik.txt oraz wyświetl w oknie graficznym,
    7. zapisz do pliku tekstowego osoby.txt podane przez użytkownika dane kilku osób (minimum trzech) takie jak: imię, nazwisko i zawód, a następnie wyświetli zawartość tego pliku w oknie graficznym,
    8. na podstawie podanych przez użytkownika parametrów współczynników funkcji kwadratowej wyznacz jej ekstremum i podaj, czy ramiona paraboli skierowane są do góry czy do dołu, wyniki zapisz do pliku tekstowego wyniki.txt oraz wyświetl w oknie graficznym,
    9. na podstawie podanych przez użytkownika parametrów współczynników funkcji kwadratowej wyznacz jej pierwiastki oraz ekstremum, wyniki zapisz do pliku tekstowego wyniki.txt oraz wyświetl w oknie graficznym,
    10. dla liczby podanej przez użytkownika wyznacz parę liczb zgodnie z hipotezą Goldbacha (każda liczba parzysta większa od dwóch jest sumą dwóch liczba pierwszych),
    11. dla zakresu liczb podanego przez użytkownika wyznacz parę liczb zgodnie z hipotezą Goldbacha (każda liczba parzysta większa od dwóch jest sumą dwóch liczba pierwszych),
    12. sprawdź, czy podany przez użytkownika ciąg znaków jest palindromem,
    13. odczytaj podany przez użytkownika znak i sprawdź, ile razy występuje w pliku tekstowym dane.txt, wynik wyświetl w oknie graficznym i zapisz w pliku wynik.txt,
    14. dla podanej przez użytkownika kwoty pieniędzy wyznacz optymalne nominały banknotów i monet, kwotę i nominały zapisz do pliku tekstowego wyniki.txt oraz wyświetl w oknie graficznym,
    15. dla podanej przez użytkownika pary liczb wyznacz NWD (największy wspólny dzielnik) oraz NWW (najmniejsza wspólna wielokrotność), podane liczby i wyniki zapisz do pliku tekstowego wyniki.txt oraz wyświetl w oknie graficznym,
    16. dla podanej przez użytkownika pary liczb wyznacz tę, której suma cyfr jest większa, wynik wyświetl w oknie graficznym i zapisz w pliku wynik.txt,
    17. pobierz od użytkownika liczby do momentu wpisania liczby 0, następnie wyznacz z nich liczby parzyste i nieparzyste, wyniki zapisz do pliku tekstowego wyniki.txt oraz wyświetl w oknie graficznym,
    18. wczytaj wyrazy z pliku tekstowego dane.txt i posortuj je alfabetycznie, wyniki zapisz do pliku tekstowego wyniki.txt oraz wyświetl w oknie graficznym,
    19. wygeneruj 10 zestawów po 6 liczb pseudolosowych z zakresu od 1 do 49, sprawdź, czy nie ma powtórzeń, czyli dwóch tych samych liczb w zestawie, w razie wykrycia powtórzeń, należy

wygenerować nowy zestaw liczb, dodatkowo każdy z zestawów uporządkuj rosnąco i zapisz do pliku tekstowego wyniki.txt oraz wyświetl w oknie graficznym,

20. wygeneruj zestaw 10 liczb naturalnych pseudolosowych z zakresu od 100 do 10000 oraz oblicz dla każdej z nich sumę jej cyfr, wyniki zapisz do pliku tekstowego wyniki.txt oraz wyświetl w oknie graficznym,
21. wygeneruj zestaw 10 liczb naturalnych pseudolosowych pięciocyfrowych oraz oblicz dla każdej z nich sumę jej cyfr, wyniki zapisz do pliku tekstowego wyniki.txt oraz wyświetl w oknie graficznym,
22. wygeneruj zestaw 10 liczb naturalnych pseudolosowych z zakresu od 100 do 10000 oraz wyznacz dla każdej z nich zestaw czynników pierwszych (iloczyn liczb pierwszych równy wylosowanej liczbie), wyniki zapisz do pliku tekstowego wyniki.txt oraz wyświetl w oknie graficznym,
23. wygeneruj zestaw 10 liczb naturalnych pseudolosowych pięciocyfrowych oraz oblicz dla każdej z nich sumę zestawu czynników pierwszych (suma liczb pierwszych, których iloczyn jest równy wylosowanej liczbie), wyniki zapisz do pliku tekstowego wyniki.txt oraz wyświetl w oknie graficznym,
24. sprawdź poprawność podawanych przez użytkownika numerów PESEL, numery PESEL oraz ich poprawność zapisz do pliku tekstowego wyniki.txt oraz wyświetl w oknie graficznym,
25. wyświetl znaki ASCII dla języka polskiego z wybranego przez użytkownika zakresu (dozwolone od 33 do 255), znaki i ich numery zapisz do pliku tekstowego wyniki.txt oraz wyświetl w oknie graficznym,
26. po wybraniu przez użytkownika dnia tygodnia wyświetl jego poprzednik i następnik, wyniki zapisz do pliku tekstowego wyniki.txt,
27. po wybraniu przez użytkownika miesiąca wyświetl jego poprzednik i następnik, wyniki zapisz do pliku tekstowego wyniki.txt,
28. narysuj w polu tekstowym ramkę o podanych przez użytkownika rozmiarach i znaku rysowania, wyniki zapisz do pliku tekstowego wyniki.txt,
29. oblicz odległość pomiędzy dwoma punktami, których współrzędne poda użytkownika, wyniki zapisz do pliku tekstowego wyniki.txt oraz wyświetl w oknie graficznym, pomoc:
    ```
    L = math.sqrt((x2 -x1)2 + (y2-y1)2)
    ```
30. po podaniu przez użytkownika daty bieżącego roku wyświetl numer tego dnia w roku kalendarzowym, wyniki zapisz do pliku tekstowego wyniki.txt,
31. po podaniu przez użytkownika daty bieżącego roku wyświetl odpowiadający mu znak zodiaku, wyniki zapisz do pliku tekstowego wyniki.txt,
32. w polu tekstowym okna graficznego zasymuluj proces formatowania dysku,
33. wyświetl w trzech kolumnach liczby nieparzyste z zakresu 1..199, wyniki zapisz do pliku tekstowego wyniki.txt,
34. wygeneruj dźwięk przypominający sygnał karetki pogotowia z możliwością wyboru trzech tonacji,
35. dla podanej przez użytkownika podstawy (zakres od 0..100) i wykładnika (zakres od 0..5) oblicz poszczególne potęgi, wyniki zapisz do pliku tekstowego wyniki.txt oraz wyświetl w oknie graficznym,
36. sprawdź, czy podane przez użytkownika dwa słowa są anagramami (ten sam zestaw znaków), wyniki zapisz do pliku tekstowego wyniki.txt oraz wyświetl w oknie graficznym,
37. zaszyfruj podany ciąg znaków przez użytkownika metodą monoalfabetyczną (jedno słowo klucz, alfabet jawny i szyfrowy – pomoc: inf. eur.zakr.rozsz.cz.1.str.197),
38. zapisz w postaci alfabetu Morsea podany przez użytkownika adres IP, wyniki zapisz do pliku tekstowego wyniki.txt oraz wyświetl w oknie graficznym.

Wszystkie pliki składowe projektu spakuj i zapisz w pliku pod nazwą **$nazwisko_$klasa_$gr_python.7z** oraz prześlij do nauczyciela w postaci załącznika na adres greszata@zs9elektronik.pl.

https://zpe.gov.pl/szukaj?query=python
http://uoo.univ.szczecin.pl/~jakubs/py/gfx.html

UWAGA!

W razie problemów kieruj pytania do nauczyciela na adres greszata@zs9elektronik.pl.