



Sprawozdanie z laboratorium

Języki skryptowe

JS 102: Hello world

19.03.2025

Prowadzący: dr Beata Zieba

Grupa: 24-INF-SP/A

Hubert Jarosz

Oliwer Pawelski

1 Wstęp

Laboratorium miało na celu zapoznanie się z podstawowymi elementami języka Python, jego składnią oraz możliwościami operowania na różnych typach danych. W ramach ćwiczeń analizowano sposób działania interpretera oraz strukturę kodu źródłowego. Przeprowadzono operacje na liczbach, tekstach i kolekcjach danych, co pozwoliło zrozumieć kluczowe mechanizmy manipulacji danymi w Pythonie.

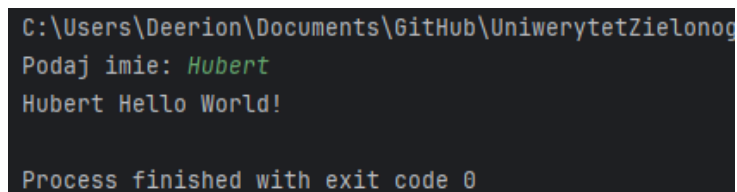
Dodatkowo zajęcia obejmowały pracę z listami i słownikami, w tym ich modyfikację oraz formatowanie i przekształcanie łańcuchów znaków. Wykorzystano również metody konwersji liczb między różnymi systemami liczbowymi. Na zakończenie omówiono fundamentalne koncepcje, takie jak typy danych, sekwencje oraz ich właściwości, co pozwoliło na lepsze zrozumienie zasad rządzących językiem Python.

2 Zadania

Zadanie 1 Napisać program wyświetlający zdanie „Hello, World!”, z zastosowaniem instrukcji `print(string)`.

```
1 imie = input("Podaj imie: ")
2 print(imie + " Hello World!")
```

Listing 1: Kod do zadania 1



```
C:\Users\Deerion\Documents\GitHub\UniwersytetZielonog
Podaj imie: Hubert
Hubert Hello World!

Process finished with exit code 0
```

Zadanie 2 Napisać program wyświetlający „Zen Pythona” w wielu wierszach.

```
1 zen = """Beautiful is better than ugly.
2 Explicit is better than implicit.
3 Simple is better than complex.
4 Complex is better than complicated.
5 Flat is better than nested.
6 Sparse is better than dense.
7 Readability counts.
8 Special cases aren't special enough to break the rules.
9 Although practicality beats purity.
10 Errors should never pass silently.
11 Unless explicitly silenced.
12 In the face of ambiguity, refuse the temptation to guess.
13 There should be one and preferably only one obvious
14 way to do it.
15 Although that way may not be obvious at first unless
16 you're Dutch.
17 Now is better than never.
18 Although never is often better than right now.
19 If the implementation is hard to explain, it's a bad idea.
20 If the implementation is easy to explain, it may be a
21 good idea.
22 Namespaces are one honking great idea let's do more
23 of those!"""
24 print(zen)
```

Listing 2: Kod do zadania 2

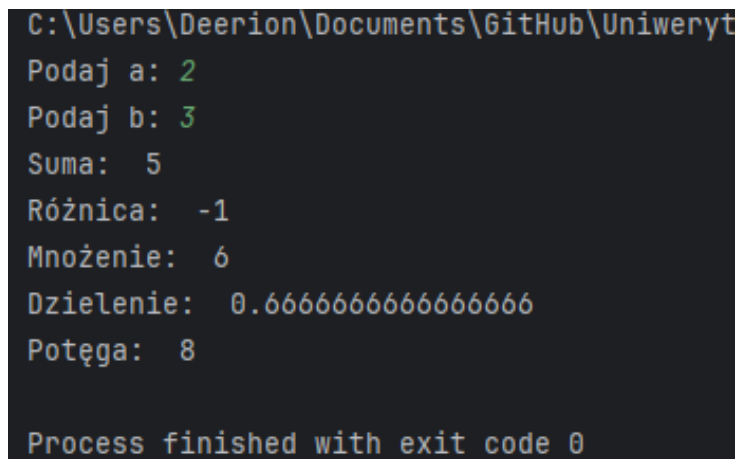
```
C:\Users\Deerion\Documents\GitHub\UniwersytetZielonogorski\Semes
Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
Readability counts.
Special cases aren't special enough to break the rules.
Although practicality beats purity.
Errors should never pass silently.
Unless explicitly silenced.
In the face of ambiguity, refuse the temptation to guess.
There should be one-and preferably only one-obvious
way to do it.
Although that way may not be obvious at first unless
you're Dutch.
Now is better than never.
Although never is often better than right now.
If the implementation is hard to explain, it's a bad idea.
If the implementation is easy to explain, it may be a
good idea.
Namespaces are one honking great idea-let's do more
of those!

Process finished with exit code 0
```

Zadanie 3 Napisać program, który wyświetli wynik dodawania, mnożenia, dzielenia, odejmowania i potęgowania dwóch liczb przypisanych do zmiennych a,b.

```
1 a = int(input("Podaj a: "))
2 b = int(input("Podaj b: "))
3
4 suma = a + b
5 roznica = a - b
6 mnozenie = a * b
7 dzielenie = a / b
8 potega = a ** b
9
10
11 print("Suma: ", suma)
12 print("Roznica: ", roznica)
13 print("Mnozenie: ", mnozenie)
14 print("Dzielenie: ", dzielenie)
15 print("Potega: ", potega)
```

Listing 3: Kod do zadania 3



```
C:\Users\Deerion\Documents\GitHub\Uniweryt
Podaj a: 2
Podaj b: 3
Suma: 5
Różnica: -1
Mnożenie: 6
Dzielenie: 0.6666666666666666
Potęga: 8

Process finished with exit code 0
```

Zadanie 4 Napisać program sklejający listy a,b,c kolejnych 10 liczb całkowitych, zespolonych i rzeczywistych (dla liczb całkowitych i zespolonych i rzeczywistych przyjąć 0 za początek listy, dla „kolejnych” liczb rzeczywistych przyjąć krok 0.1)

```
1 a = list(range(10))
2 b = [complex(i, i) for i in range(10)]
3 c = [i/10 for i in range(10)]
4
5 result = a + b + c
6 print(result)
```

Listing 4: Kod do zadania 4

```
C:\Users\sercin\Documents\GitHub\University\id10\prog\skil\Semestr_II\zadania\skryptow\Labolatorium_zadania\python.exe C:\Users\sercin\Documents\GitHub
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 0j, (1+1j), (2+2j), (3+3j), (4+4j), (5+5j), (6+6j), (7+7j), (8+8j), (9+9j), 0.0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9]
Process finished with exit code 0
```

Zadanie 5 **Napisać program, który wyświetli „Zen Pythona” wielkimi literami.**

```
1 zen = """Beautiful is better than ugly.  
2 Explicit is better than implicit.  
3 Simple is better than complex.  
4 Complex is better than complicated.  
5 Flat is better than nested.  
6 Sparse is better than dense.  
7 Readability counts.  
8 Special cases aren't special enough to break the rules.  
9 Although practicality beats purity.  
10 Errors should never pass silently.  
11 Unless explicitly silenced.  
12 In the face of ambiguity, refuse the temptation to guess.  
13 There should be one and preferably only one obvious  
14 way to do it.  
15 Although that way may not be obvious at first unless  
16 you're Dutch.  
17 Now is better than never.  
18 Although never is often better than right now.  
19 If the implementation is hard to explain, it's a bad idea.  
20 If the implementation is easy to explain, it may be a  
21 good idea.  
22 Namespaces are one honking great idea—let's do more  
23 of those!"""  
24 print(zen.upper())
```

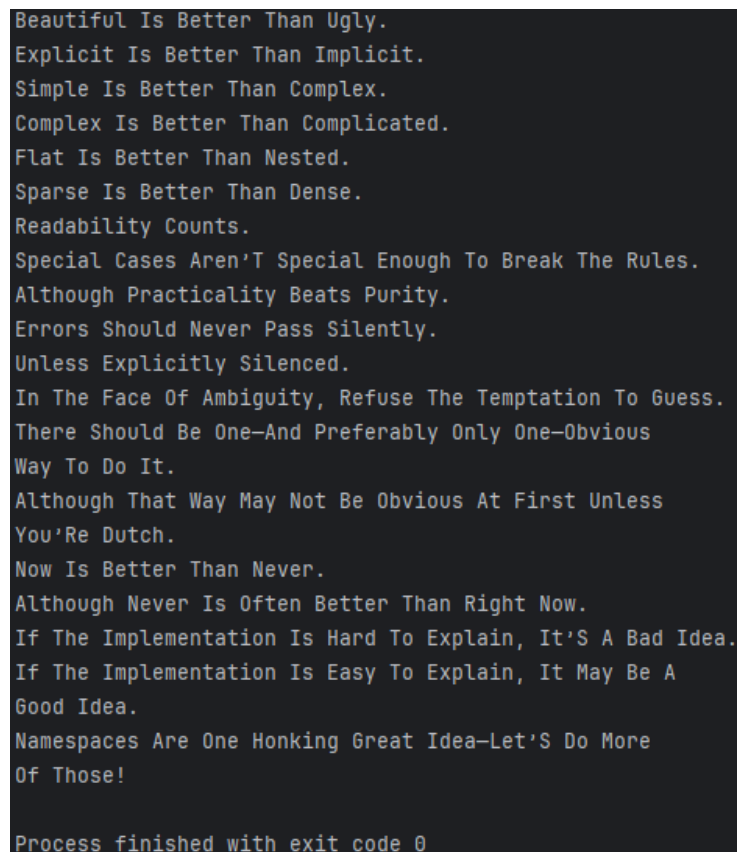
Listing 5: Kod do zadania 5

```
C:\Users\Deerion\Documents\GitHub\UniwersytetZielonogorski\Semestr_4  
BEAUTIFUL IS BETTER THAN UGLY.  
EXPLICIT IS BETTER THAN IMPLICIT.  
SIMPLE IS BETTER THAN COMPLEX.  
COMPLEX IS BETTER THAN COMPLICATED.  
FLAT IS BETTER THAN NESTED.  
SPARSE IS BETTER THAN DENSE.  
READABILITY COUNTS.  
SPECIAL CASES AREN'T SPECIAL ENOUGH TO BREAK THE RULES.  
ALTHOUGH PRACTICALITY BEATS PURITY.  
ERRORS SHOULD NEVER PASS SILENTLY.  
UNLESS EXPLICITLY SILENCED.  
IN THE FACE OF AMBIGUITY, REFUSE THE TEMPTATION TO GUESS.  
THERE SHOULD BE ONE-AND PREFERABLY ONLY ONE-OBVIOUS  
WAY TO DO IT.  
ALTHOUGH THAT WAY MAY NOT BE OBVIOUS AT FIRST UNLESS  
YOU'RE DUTCH.  
NOW IS BETTER THAN NEVER.  
ALTHOUGH NEVER IS OFTEN BETTER THAN RIGHT NOW.  
IF THE IMPLEMENTATION IS HARD TO EXPLAIN, IT'S A BAD IDEA.  
IF THE IMPLEMENTATION IS EASY TO EXPLAIN, IT MAY BE A  
GOOD IDEA.  
NAMESPACES ARE ONE HONKING GREAT IDEA—LET'S DO MORE  
OF THOSE!  
  
Process finished with exit code 0
```

Zadanie 6 Napisać program wyświetlający „Zen Pythona” każdy wyraz wielką literą.

```
1 zen = """Beautiful is better than ugly.  
2 Explicit is better than implicit.  
3 Simple is better than complex.  
4 Complex is better than complicated.  
5 Flat is better than nested.  
6 Sparse is better than dense.  
7 Readability counts.  
8 Special cases arent special enough to break the rules.  
9 Although practicality beats purity.  
10 Errors should never pass silently.  
11 Unless explicitly silenced.  
12 In the face of ambiguity, refuse the temptation to guess.  
13 There should be one and preferably only one obvious  
14 way to do it.  
15 Although that way may not be obvious at first unless  
16 you're Dutch.  
17 Now is better than never.  
18 Although never is often better than right now.  
19 If the implementation is hard to explain, it's a bad idea.  
20 If the implementation is easy to explain, it may be a  
21 good idea.  
22 Namespaces are one honking great idea lets do more  
23 of those!"""  
24 print(zen.title())
```

Listing 6: Kod do zadania 6



```
Beautiful Is Better Than Ugly.  
Explicit Is Better Than Implicit.  
Simple Is Better Than Complex.  
Complex Is Better Than Complicated.  
Flat Is Better Than Nested.  
Sparse Is Better Than Dense.  
Readability Counts.  
Special Cases Aren'T Special Enough To Break The Rules.  
Although Practicality Beats Purity.  
Errors Should Never Pass Silently.  
Unless Explicitly Silenced.  
In The Face Of Ambiguity, Refuse The Temptation To Guess.  
There Should Be One-And Preferably Only One-Obvious  
Way To Do It.  
Although That Way May Not Be Obvious At First Unless  
You'Re Dutch.  
Now Is Better Than Never.  
Although Never Is Often Better Than Right Now.  
If The Implementation Is Hard To Explain, It'S A Bad Idea.  
If The Implementation Is Easy To Explain, It May Be A  
Good Idea.  
Namespaces Are One Honking Great Idea-Let'S Do More  
Of Those!  
  
Process finished with exit code 0
```

**Zadanie 7 Napisać program usuwający wszystkie białe znaki z „Zen Pythona”.
Zastosować metodę replace(old string, new string).**

```
1 zen = """Beautiful is better than ugly.  
2 Explicit is better than implicit.  
3 Simple is better than complex.  
4 Complex is better than complicated.  
5 Flat is better than nested.  
6 Sparse is better than dense.  
7 Readability counts.  
8 Special cases arent special enough to break the rules.  
9 Although practicality beats purity.  
10 Errors should never pass silently.  
11 Unless explicitly silenced.  
12 In the face of ambiguity, refuse the temptation to guess.  
13 There should be one and preferably only one obvious  
14 way to do it.  
15 Although that way may not be obvious at first unless  
16 you're Dutch.  
17 Now is better than never.  
18 Although never is often better than right now.  
19 If the implementation is hard to explain, it's a bad idea.  
20 If the implementation is easy to explain, it may be a  
21 zen_bez_spacji = zen.replace(" ", "")  
22 print(zen_bez_spacji)
```

Listing 7: Kod do zadania 7

```
Beautifulisbetterthanugly.  
Explicitisbetterthanimplicit.  
Simpleisbetterthancomplex.  
Complexisbetterthancomplicated.  
Flatisbetterthannested.  
Sparseisbetterthandense.  
Readabilitycounts.  
Specialcasesaren'tspecialenoughtobreaktherules.  
Althoughpracticalitybeatspurity.  
Errorsshouldneverpasssilently.  
Unlessexplicitlysilenced.  
Inthefaceofambiguity,refusethetemptationtoguess.  
Thereshouldbeone-andpreferablyonlyone-obvious  
waytodoit.  
Althoughthatwaymaynotbeobviousatfirstunless  
you'reDutch.  
Nowisbetterthannever.  
Althoughneverisoftenbetterthanrightnow.  
Iftheimplementationishardtoexplain,it'sabadidea.  
Iftheimplementationiseasytoexplain,itmaybe  
goodidea.  
Namespacesareonehonkinggreatidea-let'sdomore  
ofthose!
```


Zadanie 8 Korzystając z dokumentacji Pythona dotyczącej zmiennych podstawowych napisać program wyświetlający liczbę całkowitą w postaci ósemkowej, szesnastkowej i dwójkowej.

```
1 a = 10
2 print("Liczba całkowita: ", a)
3 print("Liczba w postaci ósemkowej : ", oct(a))
4 print("Liczba w postaci szesnastkowej: ", hex(a))
5 print("Liczba w postaci dwójkowej: ", bin(a))
```

Listing 8: Kod do zadania 8

```
C:\Users\Deerion\Documents\GitHub\UniwersytetZ
Liczba całkowita: 10
Liczba w postaci ósemkowej: 0o12
Liczba w postaci szesnastkowej: 0xa
Liczba w postaci dwójkowej: 0b1010

Process finished with exit code 0
```

Zadanie 9 W jaki sposób można modyfikować łańcuch znaków?

```
1 # W jaki sposob mozna modyfikowa lancuch znakow?
2
3 # Original string
4 original_string = "Hello , World!"
5
6 # Concatenation
7 concatenated_string = original_string + " How are you?"
8
9 # Replacement
10 replaced_string = original_string.replace("World", "Python")
11
12 # Uppercase
13 uppercase_string = original_string.upper()
14
15 # Lowercase
16 lowercase_string = original_string.lower()
17
18 # Slicing
19 sliced_string = original_string[7:12]
20
21 # Displaying the results
22 print("Original String:\t", original_string)
23 print("Concatenated String:", concatenated_string)
24 print("Replaced String:\t", replaced_string)
25 print("Uppercase String:\t", uppercase_string)
26 print("Lowercase String:\t", lowercase_string)
27 print("Sliced String:\t\t", sliced_string)
```

Listing 9: Kod do zadania 9

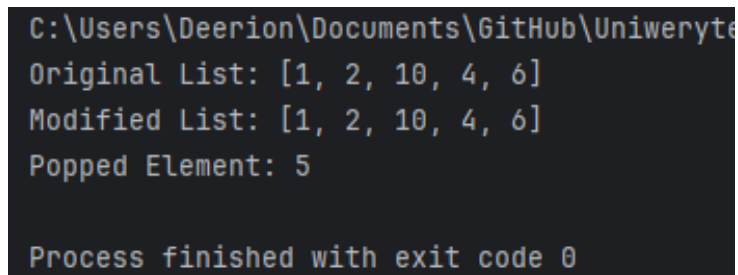
```
C:\Users\Deerion\Documents\GitHub\UniwersytetZielony>python zadanie9.py
Original String:      Hello, World!
Concatenated String: Hello, World! How are you?
Replaced String:      Hello, Python!
Uppercase String:     HELLO, WORLD!
Lowercase String:     hello, world!
Sliced String:        World

Process finished with exit code 0
```

Zadanie 10 Podać cztery operacje modyfikujące obiekt listy.

```
1 # Podac cztery operacje modyfikujace obiekt listy.
2
3 # Original list
4 original_list = [1, 2, 3, 4, 5]
5
6 # 1. Append: Adding an element to the end of the list
7 original_list.append(6)
8
9 # 2. Remove: Removing the first occurrence of an element
10 original_list.remove(3)
11
12 # 3. Insert: Inserting an element at a specific position
13 original_list.insert(2, 10)
14
15 # 4. Pop: Removing and returning an element at a specific position
16 popped_element = original_list.pop(4)
17
18 # Displaying the results
19 print("Original List:", original_list)
20 print("Modified List:", original_list)
21 print("Popped Element:", popped_element)
```

Listing 10: Kod do zadania 10



```
C:\Users\Deerion\Documents\GitHub\Uniweryte
Original List: [1, 2, 10, 4, 6]
Modified List: [1, 2, 10, 4, 6]
Popped Element: 5

Process finished with exit code 0
```

Zadanie 11 Podać cztery operacje modyfikujące obiekt słownika.

```
1 # Podac cztery operacje modyfikujace obiekt slownika.
2
3 # Original dictionary
4 original_dict = {'a': 1, 'b': 2, 'c': 3}
5
6 print("Original dictionary:", original_dict)
7
8 # 1. Add a new key-value pair
9 original_dict['d'] = 4
10
11 print("Modified Dictionary:", original_dict)
12
13 # 2. Update the value of an existing key
14 original_dict['a'] = 10
15
16 print("Modified Dictionary:", original_dict)
17
18 # 3. Remove a key-value pair using pop
19 removed_value = original_dict.pop('b')
20 print("Removed Value:", removed_value)
21 print("Modified Dictionary:", original_dict)
22
23 # 4. Clear all key-value pairs
24 original_dict.clear()
25 print("Modified Dictionary:", original_dict)
```

Listing 11: Kod do zadania 11

```
C:\Users\Deerion\Documents\GitHub\UniwersytetZielonogorski>python3 zad11.py
Original dictionary: {'a': 1, 'b': 2, 'c': 3}
Modified Dictionary: {'a': 1, 'b': 2, 'c': 3, 'd': 4}
Modified Dictionary: {'a': 10, 'b': 2, 'c': 3, 'd': 4}
Removed Value: 2
Modified Dictionary: {'a': 10, 'c': 3, 'd': 4}
Modified Dictionary: {}

Process finished with exit code 0
```

Zadanie 12 Co to jest interpreter Pythona?

```
1 # Co to jest interpreter Pythona?
2
3 def explain_python_interpreter():
4     explanation = """
5         Interpreter Pythona to program, który wykonuje kod napisany w
6         języku Python.
7         Działa on poprzez analizowanie i wykonywanie instrukcji zawartych
8         w kodzie źródłowym.
9
10        Python jest językiem interpretowanym, co oznacza, że jego kod
11        jest wykonywany linia po linii,
12        a nie kompilowany do pliku binarnego przed uruchomieniem. Dzięki
13        temu umożliwia szybkie testowanie
14        i eksperymentowanie z kodem.
15
16        Interpreter Pythona może działać w dwóch trybach:
17        1. Tryb interaktywny      użytkownik wpisuje polecenia
18        bezpośrednio i otrzymuje natychmiastowe wyniki.
19        Przykład:
20        $ python
21        >>> print("Hello, World!")
22        Hello, World!
23
24        2. Tryb wsadowy          interpreter wykonuje skrypty zapisane w
25        plikach:
26        $ python script.py
27        """
28     print(explanation)
29
30 explain_python_interpreter()
```

Listing 12: Kod do zadania 12

Zadanie 13 Co to jest kod źródłowy?

```
1 # Co to jest kod zdrojowy?
2
3 def explain_source_code():
4     explanation = """
5     Kod zdrojowy to zbior kodu napisanego w jezyku programowania
6     zrozumialym dla czlowieka.
7     Jest to oryginalny kod, ktory programista pisze, a ktory
8     nastepnie jest kompilowany lub interpretowany
9     w celu stworzenia programu wykonywalnego. Kod zdrojowy jest
10    niezbedny do zrozumienia, utrzymania
11    i modyfikacji oprogramowania.
12
13    Przyklad prostego kodu zdrojowego w Pythonie:
14    def greet():
15        print("Hello, World!")
16
17    greet()
18    """
19    print(explanation)
20 explain_source_code()
```

Listing 13: Kod do zadania 13

Zadanie 14 Podać cztery podstawowe (wbudowane) typy danych Pythona

```
1 # Podac cztery podstawowe (wbudowane) typy danych Pythona
2
3 def explain_basic_data_types():
4     explanation = """
5     Cztery podstawowe (wbudowane) typy danych w Pythonie to:
6
7     1. int (liczby calkowite) – reprezentuje liczby calkowite, np. 1,
8       2, 3.
9     2. float (liczby zmiennoprzecinkowe) – reprezentuje liczby
10    rzeczywiste, np. 1.0, 2.5, 3.14.
11    3. str (lancuchy znakow) – reprezentuje tekst, np. "Hello, World
12    !".
13    4. bool (wartosci logiczne) – reprezentuje wartosci logiczne, np.
14    True, False.
15
16    Przyklady uzycia:
17    a = 10 # int
18    b = 3.14 # float
19    c = "Hello, World!" # str
20    d = True # bool
21
22    print(f"a: {a}, typ: {type(a)}")
23    print(f"b: {b}, typ: {type(b)}")
24    print(f"c: {c}, typ: {type(c)}")
25    print(f"d: {d}, typ: {type(d)}")
26    """
27
28    print(explanation)
29
30 explain_basic_data_types()
```

Listing 14: Kod do zadania 14

Zadanie 15 Dlaczego te typy nazywane są podstawowymi (wbudowanymi)?

```
1 # Dlaczego te typy nazywane sa podstawowymi (wbudowanymi)?
2
3 def explain_why_basic_data_types():
4     explanation = """
5     Te typy danych nazywane sa podstawowymi (wbudowanymi), poniewaz
6     sa integralna czescia jezyka Python.
7     Sa one dostepne bez potrzeby importowania dodatkowych modutow i
8     sa fundamentalne dla wiekszosci operacji
9     wykonywanych w Pythonie. Dzieki nim mozna wykonywac podstawowe
10    operacje arytmetyczne, manipulowac tekstem,
11    oraz przechowywac wartosci logiczne.
12
13    Przyklady uzycia:
14    a = 10 # int
15    b = 3.14 # float
16    c = "Hello, World!" # str
17    d = True # bool
18
19    print(f"a: {a}, typ: {type(a)}")
20    print(f"b: {b}, typ: {type(b)}")
21    print(f"c: {c}, typ: {type(c)}")
22    print(f"d: {d}, typ: {type(d)}")
23    """
24    print(explanation)
25
26 explain_why_basic_data_types()
```

Listing 15: Kod do zadania 15

Zadanie 16 Czym jest sekwencja?

```
1 # Czym jest sekwencja?
2
3 def explain_sequence():
4     explanation = """
5     Sekwencja to uporządkowana kolekcja elementów, które mogą być
6     indeksowane i iterowane.
7     W Pythonie sekwencje obejmują typy takie jak listy, krotki,
8     łańcuchy znaków i zakresy.
9     Sekwencje pozwalają na dostęp do elementów za pomocą indeksów
10    oraz na wykonywanie operacji
11    takich jak cięcie, konkatenaacja i iteracja.
12
13    Przykłady sekwencji w Pythonie:
14    lista = [1, 2, 3, 4, 5] # lista
15    krotka = (1, 2, 3, 4, 5) # krotka
16    łańcuch = "Hello, World!" # łańcuch znaków
17    zakres = range(5) # zakres
18
19    print(f"Lista: {lista}")
20    print(f"Krotka: {krotka}")
21    print(f"Łańcuch: {łańcuch}")
22    print(f"Zakres: {list(zakres)}")
23    """
24    print(explanation)
25
26 explain_sequence()
```

Listing 16: Kod do zadania 16

Zadanie 17 Podać dwa typy niezmiennicze.

```
1 # Podac dwa typy niezmiennicze.
2
3 def explain_immutable_types():
4     explanation = """
5     Dwa typy niezmiennicze w Pythonie to:
6
7     1. tuple (krotka) – jest to uporządkowana kolekcja elementów,
8     która jest niezmienna, co oznacza, że
9     po utworzeniu krotki nie można zmienić jej elementów.
10    2. str (łańcuch znaków) – jest to sekwencja znaków, która jest
11    niezmienna, co oznacza, że
12    po utworzeniu łańcucha znaków nie można zmienić jego
13    zawartości.
14
15    Przykłady użycia:
16    krotka = (1, 2, 3)
17    lancuch = "Hello, World!"
18
19    print(f"Krotka: {krotka}, typ: {type(krotka)}")
20    print(f"Lancuch: {lancuch}, typ: {type(lancuch)}")
21    """
22    print(explanation)
23
24 explain_immutable_types()
```

Listing 17: Kod do zadania 17