
	<p>Politechnika Bydgoska im. J. J. Śniadeckich  Wydział Telekomunikacji,  Informatyki i Elektrotechniki  Zakład Systemów Teleinformatycznych</p>			
<b>Przedmiot</b>	Skryptowe języki programowania			
<b>Prowadzący</b>	mgr inż. Martyna Tarczewska			
<b>Temat</b>	Obiektowość Pythona			
<b>Student</b>	Paweł Jońca			
<b>Nr lab.</b>	4	<b>Data wykonania</b>	31.10	
<b>Ocena</b>		<b>Data oddania spr.</b>	31.10	

## Zad 1

```

lab04zad1.py
1  from typing import List #importuje List z modułu typing aby określić typ listy przechowujący oceny marks
2  class Student: #tworzę klasę Student 2usages
3      def __init__(self): #konstruktor __init__
4          self.name = ""
5          self.last_name = ""
6          self.marks = []
7          self.index_number = None
8      def give_name(self, name: str, last_name: str) -> None: #metoda ustawia imię i nazwisko studenta i przyjmuje dwa argumenty name i last_name, zapisuje je w obiekcie 1usage
9          self.name = name
10         self.last_name = last_name
11     def give_mark(self, mark: int) -> None: #metoda pozwala dodać ocenę do listy ocen (marks) 2usages
12         #Argument mark jest dodawany do listy
13         self.marks.append(mark)
14     def get_marks(self) -> List[int]: return self.marks #zwraca listę wszystkich ocen 1usage
15     def calculate_average(self) -> float: #metoda oblicza średnią ocen, jeśli lista jest pusta zwraca 0.0 1usage
16         if not self.marks:
17             return 0.0
18         return sum(self.marks)/len(self.marks)
19     def assign_index_number(self, index_number: int) -> None: #metoda przypisuje numer indeksu 1usage
20         self.index_number = index_number
21     def say_hello(self) -> None: 1usage
22         print(f"Hello! I'm {self.name} {self.last_name}, my index number is {self.index_number}")
23
24 s = Student() # Tworzymy obiekt studenta
25 s.give_name(name="Paul", last_name="Jonnes") # Ustawiamy imię i nazwisko na "Jane Doe"
26 s.assign_index_number(123456) # Przypisujemy numer indeksu 123456
27 s.give_mark(5) # Dodajemy ocenę 5 do listy ocen
28 Student.give_mark(s, mark=3) # Dodajemy ocenę 3 w alternatywny sposób (bezpośrednio wywołując funkcję na klasie)
29
30 print(s.get_marks()) # Wyświetla listę ocen: [5, 3]
31 print("Average mark:", s.calculate_average()) # Wyświetla średnią ocen: 4.0
32 s.say_hello() # Wyświetla powitanie z numerem indeksu
33
Run lab04zad1
"C:\Program Files\Python312\python.exe" "C:\Users\pawel\Documents\03 - studia\Skryptowe Języki programowania\pythonProject1\lab04zad1.py"
[5, 3]
Average mark: 4.0
Hello! I'm Paul Jonnes, my index number is 123456

```

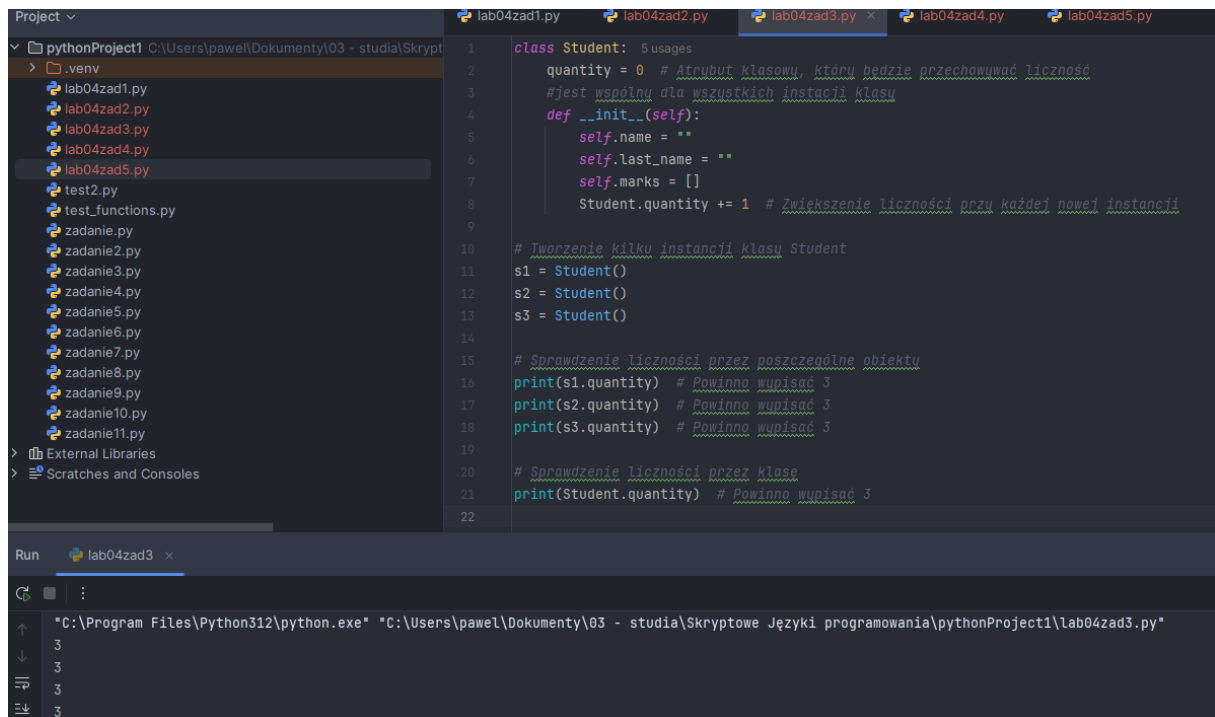
## Zad 2

```
1 class Vehicle: 2 usages
2     def get_sound(self): 1 usage
3         print("vehicle's brum brum")
4
5     def get_owner(self) -> str: #już dodałem metode i działa 1 usage
6         return "Unknown owner"
7
8 class Car(Vehicle): 1 usage
9     def __init__(self, owner: str, table: str):
10         self.owner = owner
11         self.table = table
12
13     def get_sound(self): 1 usage
14         print("car's brum brum")
15
16     def get_owner(self) -> str: 1 usage
17         return self.owner
18
19 # Tworzenie instancji klas
20 vehicle_instance = Vehicle()
21 car_instance = Car(owner="Paweł", table="XYZ 1234")
22
23 # Wywołanie metody get_sound
24 vehicle_instance.get_sound() # Powinno wypisać "vehicle's brum brum"
25 car_instance.get_sound()    # Powinno wypisać "car's brum brum"
26
27 # Wywołanie metody get_owner
28 try:
29     print(car_instance.get_owner()) # Powinno wypisać właściciela ("Paweł")
30     print(vehicle_instance.get_owner()) # To spowoduje błąd, bo Vehicle nie ma metody get_owner(już ma bo dodałem)
31 except AttributeError as e:
32     print(f"Błąd: {e}")
```

```
"C:\Program Files\Python312\python.exe" "
vehicle's brum brum
car's brum brum
Paweł
Unknown owner

Process finished with exit code 0
```

### Zad 3



The screenshot shows an IDE with a project named 'pythonProject1'. The file explorer on the left lists several files, including 'lab04zad1.py' through 'lab04zad5.py'. The main editor displays the code for 'lab04zad3.py', which defines a 'Student' class. The class has a class attribute 'quantity' set to 0. The '\_\_init\_\_' method initializes instance attributes 'name', 'last\_name', and 'marks', and increments the class attribute 'quantity' by 1. The code also creates three instances of the 'Student' class (s1, s2, s3) and prints their 'quantity' values, which should all be 3. The output window at the bottom shows the execution of 'python.exe' and the output of the program, which is three '3's, one for each instance.

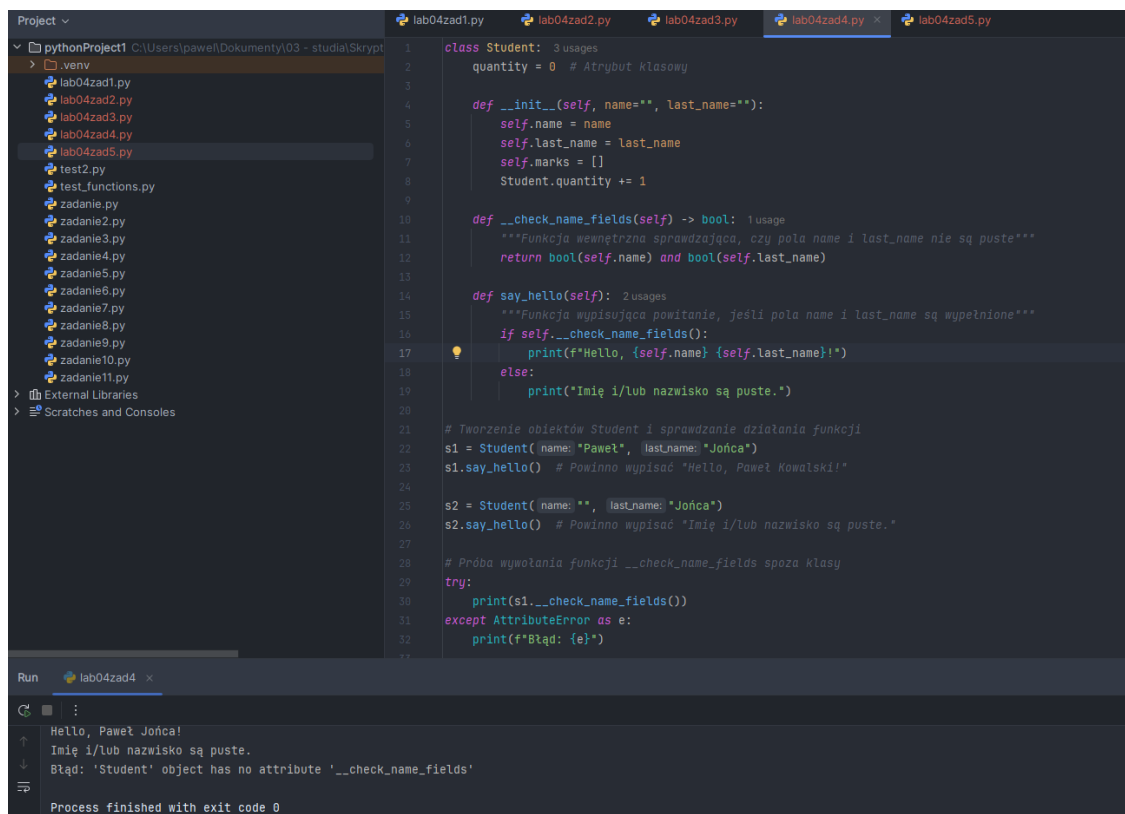
```
1 class Student:
2     quantity = 0 # Atrybut klasowy, który będzie przechowywać licznosc
3     #jest wspólny dla wszystkich instancji klasy
4     def __init__(self):
5         self.name = ""
6         self.last_name = ""
7         self.marks = []
8         Student.quantity += 1 # Zwiększenie licznosci przy każdej nowej instancji
9
10    # Tworzenie kilku instancji klasy Student
11    s1 = Student()
12    s2 = Student()
13    s3 = Student()
14
15    # Sprawdzenie licznosci przez poszczegolne obiekty
16    print(s1.quantity) # Powinno wypisac 3
17    print(s2.quantity) # Powinno wypisac 3
18    print(s3.quantity) # Powinno wypisac 3
19
20    # Sprawdzenie licznosci przez klase
21    print(Student.quantity) # Powinno wypisac 3
22
```

Run lab04zad3 x

"C:\Program Files\Python312\python.exe" "C:\Users\pawel\Dokumenty\03 - studia\Skryptowe Języki programowania\pythonProject1\lab04zad3.py"

3  
3  
3

### Zad 4



The screenshot shows an IDE with a project named 'pythonProject1'. The file explorer on the left lists several files, including 'lab04zad1.py' through 'lab04zad5.py'. The main editor displays the code for 'lab04zad4.py', which defines a 'Student' class. The class has a class attribute 'quantity' set to 0. The '\_\_init\_\_' method initializes instance attributes 'name', 'last\_name', and 'marks', and increments the class attribute 'quantity' by 1. The class also has a private method '\_\_check\_name\_fields' that checks if the 'name' and 'last\_name' attributes are non-empty. The 'say\_hello' method calls '\_\_check\_name\_fields' and prints a message. The code also creates two instances of the 'Student' class (s1, s2) and calls the 'say\_hello' method. The output window at the bottom shows the execution of 'python.exe' and the output of the program, which is 'Hello, Paweł Jońca!' and 'Imię i/lub nazwisko są puste.' followed by an error message: 'Błąd: 'Student' object has no attribute '\_\_check\_name\_fields''. The process finished with exit code 0.

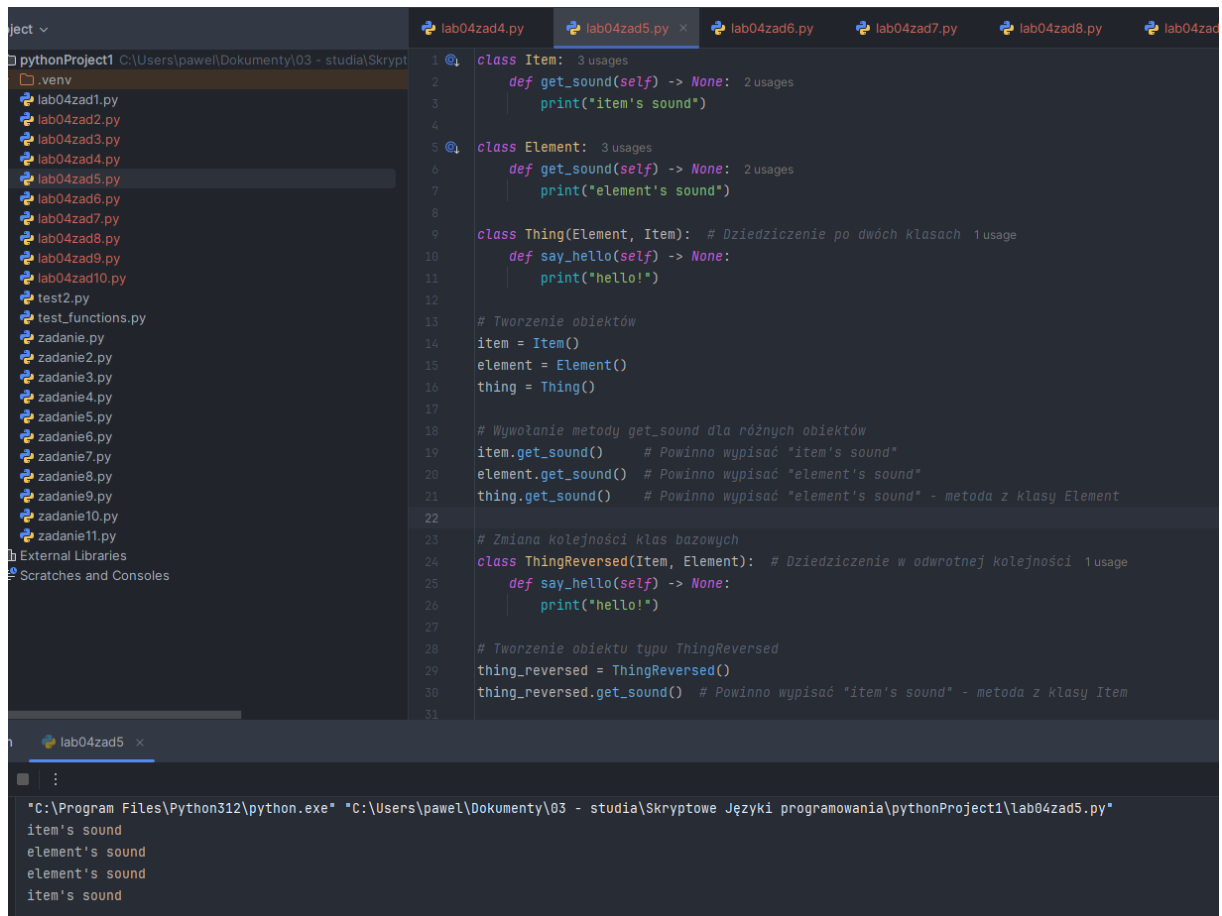
```
1 class Student:
2     quantity = 0 # Atrybut klasowy
3
4     def __init__(self, name="", last_name=""):
5         self.name = name
6         self.last_name = last_name
7         self.marks = []
8         Student.quantity += 1
9
10    def __check_name_fields(self) -> bool:
11        """Funkcja wewnętrzna sprawdzająca, czy pola name i last_name nie są puste"""
12        return bool(self.name) and bool(self.last_name)
13
14    def say_hello(self):
15        """Funkcja wypisująca powitanie, jeśli pola name i last_name są wypełnione"""
16        if self.__check_name_fields():
17            print(f"Hello, {self.name} {self.last_name}!")
18        else:
19            print("Imię i/lub nazwisko są puste.")
20
21    # Tworzenie obiektów Student i sprawdzanie działania funkcji
22    s1 = Student(name="Paweł", last_name="Jońca")
23    s1.say_hello() # Powinno wypisać "Hello, Paweł Kowalski!"
24
25    s2 = Student(name="", last_name="Jońca")
26    s2.say_hello() # Powinno wypisać "Imię i/lub nazwisko są puste."
27
28    # Próba wywołania funkcji __check_name_fields spoza klasy
29    try:
30        print(s1.__check_name_fields())
31    except AttributeError as e:
32        print(f"Błąd: {e}")
33
```

Run lab04zad4 x

Hello, Paweł Jońca!  
Imię i/lub nazwisko są puste.  
Błąd: 'Student' object has no attribute '\_\_check\_name\_fields'  
Process finished with exit code 0

Nazwa metody `__check_name_fields` zaczyna się od podwójnego podkreślenia co sugeruje, że jest ona prywatna wewnętrzna i nie powinna być wywołana bezpośrednio spoza klasy. Kiedy próbuje to zrobić python zgłasza błąd

## Zad5



The screenshot shows a Python IDE with a file explorer on the left, a code editor in the center, and a console at the bottom. The file explorer lists several files, including lab04zad1.py through lab04zad11.py. The code editor displays the following Python code:

```
1 class Item: 3 usages
2     def get_sound(self) -> None: 2 usages
3         print("item's sound")
4
5 class Element: 3 usages
6     def get_sound(self) -> None: 2 usages
7         print("element's sound")
8
9 class Thing(Element, Item): # Dziedziczenie po dwóch klasach 1 usage
10     def say_hello(self) -> None:
11         print("hello!")
12
13 # Tworzenie obiektów
14 item = Item()
15 element = Element()
16 thing = Thing()
17
18 # Wywołanie metody get_sound dla różnych obiektów
19 item.get_sound() # Powinno wypisać "item's sound"
20 element.get_sound() # Powinno wypisać "element's sound"
21 thing.get_sound() # Powinno wypisać "element's sound" - metoda z klasy Element
22
23 # Zmiana kolejności klas bazowych
24 class ThingReversed(Item, Element): # Dziedziczenie w odwrotnej kolejności 1 usage
25     def say_hello(self) -> None:
26         print("hello!")
27
28 # Tworzenie obiektu typu ThingReversed
29 thing_reversed = ThingReversed()
30 thing_reversed.get_sound() # Powinno wypisać "item's sound" - metoda z klasy Item
31
```

The console at the bottom shows the output of the code execution:

```
"C:\Program Files\Python312\python.exe" "C:\Users\pawel\Dokumenty\03 - studia\Skryptowe Języki programowania\pythonProject1\lab04zad5.py"
item's sound
element's sound
element's sound
item's sound
```

Dziedziczenie w klasie Thing (Element, Item) przez taki zapis najpierw bierzemy Element bo jest pierwszą klasą bazową dlatego właśnie wywołanie thing.get\_sound() wypisuje element's sound

## Zad 6

```
lab04zad4.py lab04zad5.py lab04zad6.py x lab04zad7.py lab04zad8.py lab04zad9.py lab04zad10.py
1 class Student: 4 usages
2     quantity = 0
3
4     def __init__(self, name="", last_name="", marks=None):
5         self.name = name
6         self.last_name = last_name
7         self.marks = marks if marks is not None else []
8         Student.quantity += 1
9
10    def average(self) -> float: 12 usages (6 dynamic)
11        #Zwraca średnią ocen
12        return sum(self.marks) / len(self.marks) if self.marks else 0
13
14    def __lt__(self, other) -> bool:
15        return self.average() < other.average()
16
17    def __gt__(self, other) -> bool:
18        return self.average() > other.average()
19
20    def __eq__(self, other) -> bool:
21        return self.average() == other.average()
22
23    def __le__(self, other) -> bool:
24        return self.average() <= other.average()
25
26    def __ge__(self, other) -> bool:
27        return self.average() >= other.average()
28
29    def __ne__(self, other) -> bool:
30        return self.average() != other.average()
31
32    # Tworzenie obiektów Student z różnymi ocenami
33    s1 = Student(name="Paweł", last_name="Jońca", marks=[3, 4, 5])
34    s2 = Student(name="Szymon", last_name="Stawarz", marks=[5, 5, 5])
35    s3 = Student(name="Jakub", last_name="Smolarek", marks=[3, 3, 3])
36    # Testowanie operatorów porównania
37    print(s1 < s2) # True, bo średnia s1 (4.0) jest mniejsza niż średnia s2 (5.0)
38    print(s1 > s3) # True, bo średnia s1 (4.0) jest większa niż średnia s3 (3.0)
39    print(s1 == s2) # False, bo średnie są różne
40    print(s1 <= s3) # False, bo średnia s1 (4.0) nie jest mniejsza ani równa średniej s3 (3.0)
41    print(s1 >= s2) # False, bo średnia s1 (4.0) nie jest większa ani równa średniej s2 (5.0)
42    print(s1 != s3) # True, bo średnie są różne
```

```
"C:\Program
True
True
False
False
False
True
```

## Zad 7

### Bez przeciążenia

```
class Student: 6 usages
    def __init__(self, name: str, last_name: str, index: int):
        self.name = name
        self.last_name = last_name
        self.index = index

    def __eq__(self, o: 'Student') -> bool:
        return self.index == o.index

    def __ne__(self, o: 'Student') -> bool:
        return self.index != o.index

    def __lt__(self, o: 'Student') -> bool:
        return self.index < o.index

    def __gt__(self, o: 'Student') -> bool:
        return self.index > o.index

# Tworzenie instancji Student
s1 = Student( name: 'Joe', last_name: 'Doe', index: 111111)
s2 = Student( name: 'Jane', last_name: 'Key', index: 222222)

# Wywołanie repr() i str() bez przeciążania
print(repr(s1)) # Wyświetli domyślną reprezentację obiektu, np. <__main__.Student object at 0x7f96bc1a7be0>
print(str(s1)) # Wyświetli to samo, co repr(), jeśli nie ma __str__
```

```
"C:\Program Files\Python312\python.exe" "C:\Users\
<__main__.Student object at 0x000002169D2E4B90>
<__main__.Student object at 0x000002169D2E4B90>

Process finished with exit code 0
```

Z przeciążeniem

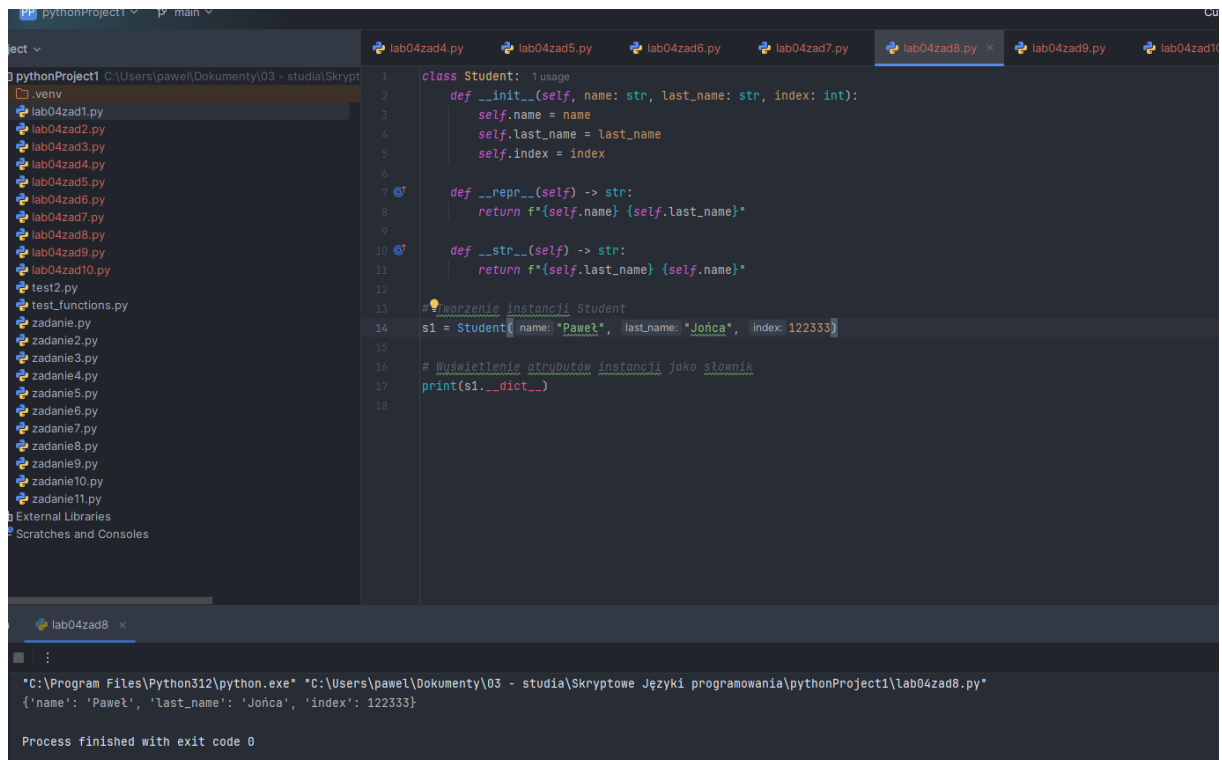
```
29 class Student: 6 usages
30     def __init__(self, name: str, last_name: str, index: int):
31         self.name = name
32         self.last_name = last_name
33         self.index = index
34
35     def __repr__(self) -> str:
36         return f'{self.name} {self.last_name}'
37
38     def __str__(self) -> str:
39         return f'{self.last_name} {self.name}'
40
41     def __eq__(self, o: 'Student') -> bool:
42         return self.index == o.index
43
44     def __ne__(self, o: 'Student') -> bool:
45         return self.index != o.index
46
47     def __lt__(self, o: 'Student') -> bool:
48         return self.index < o.index
49
50     def __gt__(self, o: 'Student') -> bool:
51         return self.index > o.index
52
53     # Tworzenie instancji Student
54     s1 = Student( name: 'Joe', last_name: 'Doe', index: 111111)
55     s2 = Student( name: 'Jane', last_name: 'Key', index: 222222)
56
57     # Wywołanie repr() i str() po przeciążeniu
58     print(repr(s1)) # Wyświetli "Joe Doe" z __repr__
59     print(str(s1)) # Wyświetli "Doe Joe" z __str__
60
```

```
"C:\Program Files\Python312\python.
Joe Doe
Doe Joe
5
4 Process finished with exit code 0
6
7
```

Bez przeciążenia `__repr__` i `__str__` Obiekt klasy `Student` jest wyświetlany w formacie domyślnym dla Pythona pokazując typ obiektu oraz jego adres w pamięci

Z przeciążeniem obiekt klasy `Student` wyświetla teraz bardziej czytelne tekstowe reprezentacje

## Zad 8



The screenshot shows a VS Code editor with a Python project named 'pythonProject1'. The file explorer on the left lists various files, including 'lab04zad1.py' through 'lab04zad11.py'. The main editor displays the code for 'lab04zad8.py'.

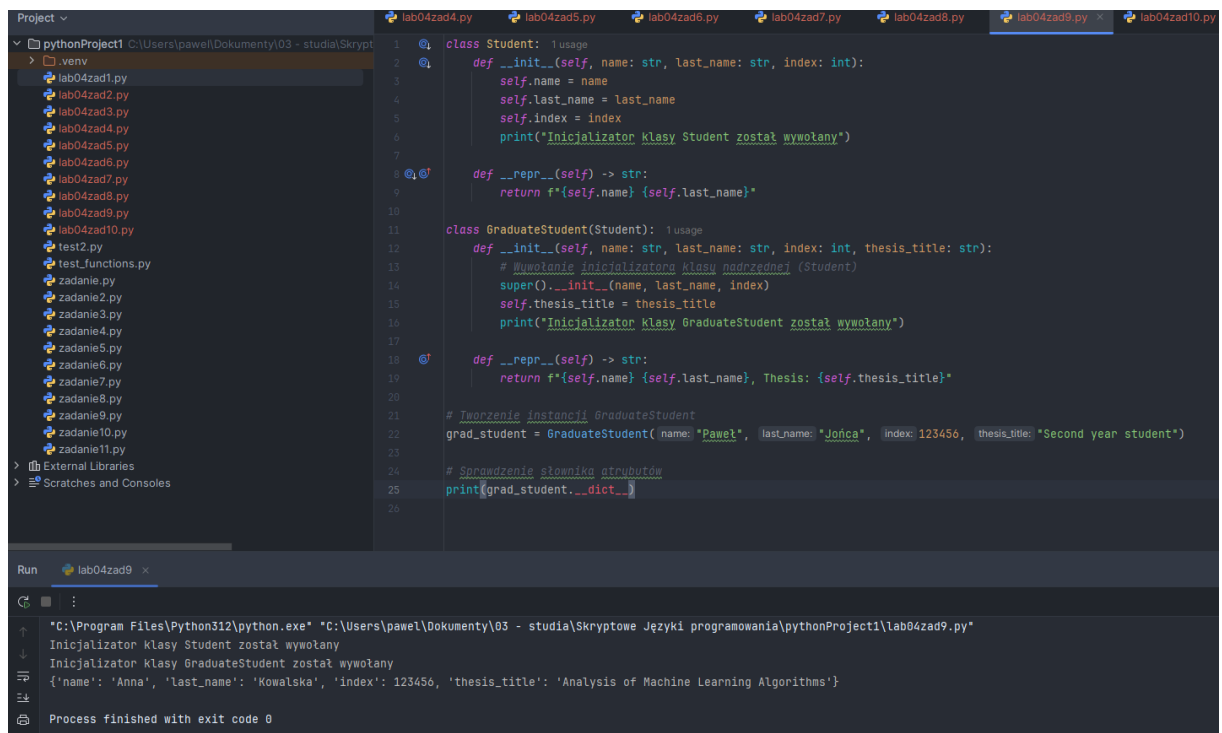
```
1 class Student: 1 usage
2     def __init__(self, name: str, last_name: str, index: int):
3         self.name = name
4         self.last_name = last_name
5         self.index = index
6
7     def __repr__(self) -> str:
8         return f'{self.name} {self.last_name}'
9
10    def __str__(self) -> str:
11        return f'{self.last_name} {self.name}'
12
13    # Tworzenie instancji Student
14    s1 = Student(name="Paweł", last_name="Jońca", index=122333)
15
16    # Wyświetlenie atrybutów instancji jako słownik
17    print(s1.__dict__)
18
```

The output window at the bottom shows the execution of the code:

```
"C:\Program Files\Python312\python.exe" "C:\Users\pawel\Dokumenty\03 - studia\Skryptowe Języki programowania\pythonProject1\lab04zad8.py"
{'name': 'Paweł', 'last_name': 'Jońca', 'index': 122333}

Process finished with exit code 0
```

## Zad 9



The screenshot shows a VS Code editor with a Python project named 'pythonProject1'. The file explorer on the left lists various files, including 'lab04zad1.py' through 'lab04zad11.py'. The main editor displays the code for 'lab04zad9.py'.

```
1 class Student: 1 usage
2     def __init__(self, name: str, last_name: str, index: int):
3         self.name = name
4         self.last_name = last_name
5         self.index = index
6         print("Inicjalizator klasy Student został wywołany")
7
8     def __repr__(self) -> str:
9         return f'{self.name} {self.last_name}'
10
11 class GraduateStudent(Student): 1 usage
12     def __init__(self, name: str, last_name: str, index: int, thesis_title: str):
13         # Wywołanie inicjalizatora klasy nadrzędnej (Student)
14         super().__init__(name, last_name, index)
15         self.thesis_title = thesis_title
16         print("Inicjalizator klasy GraduateStudent został wywołany")
17
18     def __repr__(self) -> str:
19         return f'{self.name} {self.last_name}, Thesis: {self.thesis_title}'
20
21 # Tworzenie instancji GraduateStudent
22 grad_student = GraduateStudent(name="Paweł", last_name="Jońca", index=123456, thesis_title="Second year student")
23
24 # Sprawdzenie słownika atrybutów
25 print(grad_student.__dict__)
26
```

The output window at the bottom shows the execution of the code:

```
"C:\Program Files\Python312\python.exe" "C:\Users\pawel\Dokumenty\03 - studia\Skryptowe Języki programowania\pythonProject1\lab04zad9.py"
Inicjalizator klasy Student został wywołany
Inicjalizator klasy GraduateStudent został wywołany
{'name': 'Anna', 'last_name': 'Kowalska', 'index': 123456, 'thesis_title': 'Analysis of Machine Learning Algorithms'}

Process finished with exit code 0
```



## Zad 10

```
1 class Student: 7 usages  Pawelgabrieljonca *
2     def __init__(self, name: str, last_name: str, index: int):  Pawelgabrieljonca
3         self.name = name
4         self.last_name = last_name
5         self.index = index
6
7     def __eq__(self, other) -> bool:  Pawelgabrieljonca *
8         if not isinstance(other, Student):
9             return False
10        return self.index == other.index
```

```
studia\Skrypt 1 import pytest
2 from lab04zad10 import Student
3
4 def test_eq_same_index():  Pawelgabrieljonca
5     s1 = Student(name="Anna", last_name="Kowalska", index=123456)
6     s2 = Student(name="John", last_name="Doe", index=123456)
7     assert s1 == s2 # Test sprawdza, czy studenci z tym samym indexem są równi
8
9 def test_eq_different_index():  Pawelgabrieljonca
10    s1 = Student(name="Anna", last_name="Kowalska", index=123456)
11    s2 = Student(name="John", last_name="Doe", index=654321)
12    assert s1 != s2 # Test sprawdza, czy studenci z różnymi indeksami nie są równi
13
14 def test_eq_type_check():  Pawelgabrieljonca
15    s1 = Student(name="Anna", last_name="Kowalska", index=123456)
16    assert s1 != 123456 # Test sprawdza, czy porównanie studenta z liczbą zwraca False
17
```

```
ms ✓ Tests passed: 3 of 3 tests - 0ms
"C:\Program Files\Python312\python.exe" "C:/Users/pawel/AppData/Local/Programs/PyCharm Professional/plugins/python
Testing started at 12:28 ...
Launching pytest with arguments C:\Users\pawel\Dokumenty\03 - studia\Skryptowe Języki programowania\pythonProject1

===== test session starts =====
collecting ... collected 3 items

test_student.py::test_eq_same_index PASSED [ 33%]
test_student.py::test_eq_different_index PASSED [ 66%]
test_student.py::test_eq_type_check PASSED [100%]

===== 3 passed in 0.04s =====

Process finished with exit code 0
```

Wnioski: Zadania pokazały jak korzystać z obiektowości pythona. Zadania wymagały użycia i przetestowania dziedziczenia, przeciążania oraz atrybutu `__dict__` który pozwala na łatwe sprawdzenie stanu obiektów. To wszystko pokazuje jak można rozszerzać funkcjonalność klas w pythonie.