

# PyStart #18 Debugowanie aplikacji

## Na czym polega?



- Staramy się zrozumieć, dlaczego działa lub znaleźć błąd w kodzie.
- Przechodzimy wiersz po wierszu po programie lub od punktu zatrzymania (breakpointu) do punktu zatrzymania.
- Możemy wywoływać również własne dodatkowe komendy, aby zmienić działanie programu lub by coś dodatkowo

# PyStart #18 Debugowanie aplikacji

## Metoda gumowej kaczki



# PyStart #18 Debugowanie aplikacji

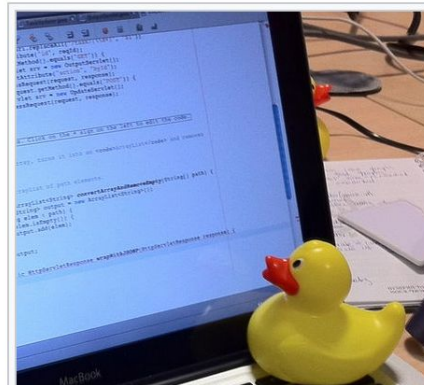
## Metoda gumowej kaczki

### Metoda gumowej kaczuszki [\[edytuj\]](#)

**Metoda gumowej kaczuszki**<sup>[1][2]</sup> – nieformalny sposób debugowania kodu.

Metoda polega na tym, że programista, próbując znaleźć błędy w kodzie (inspekcja kodu), trzyma w pobliżu gumową kaczuszkę lub inny przedmiot nieożywiony. Linia po linii, programista tłumaczy kaczuszcze lub innemu obiektowi przewidywane funkcje każdego segmentu kodu – podczas sprawdzania powinny wyjść na jaw błędy stworzonej aplikacji.

Metoda jest wersją metody „myślenia na głos”, procedury uznanej za skuteczny sposób na przyspieszenie rozwiązywania problemów<sup>[3]</sup>.



# PyStart #18 Debugowanie aplikacji

## Jak debugować aplikacje?

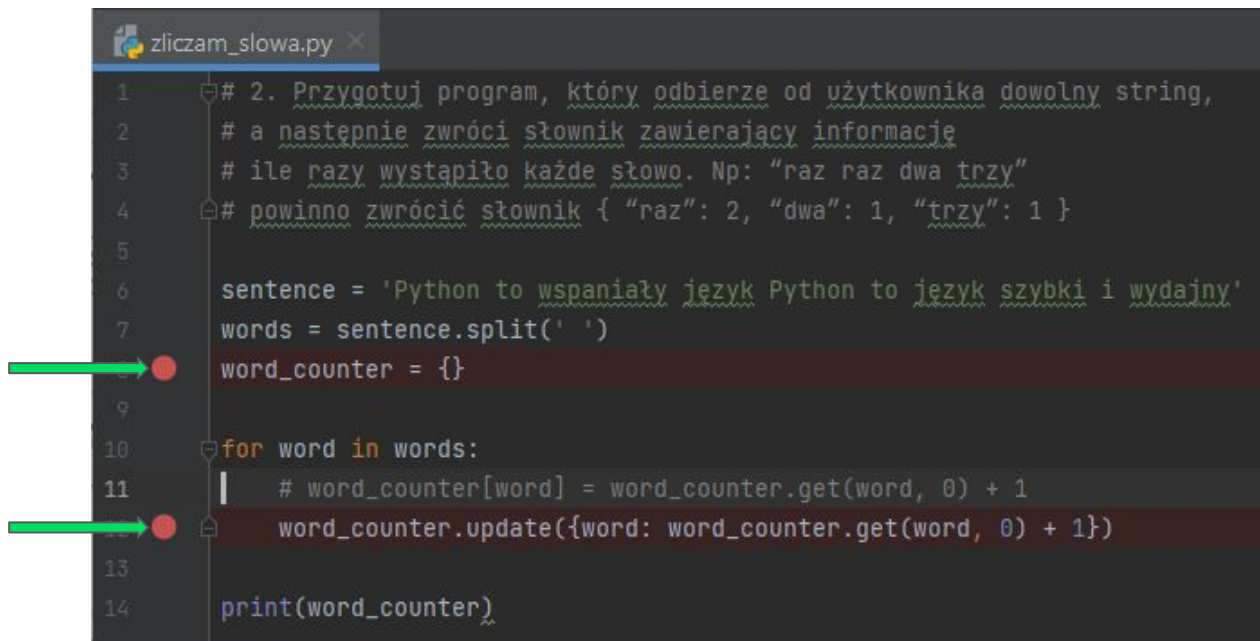


- Najprościej za pomocą `print()` jednak jest to mało wygodne.
- Dużo wygodniej za pomocą debugger'a
- Jakich rodzajów błędów szukamy?
  - ◆ błędy działania (np. logiczne)
  - ◆ przechwytywanie wyjątków

# PyStart #18 Debugowanie aplikacji

## Dobra, jak to zrobić?

→ Po pierwsze dodajemy breakpoints klikając w przestrzeń po lewej



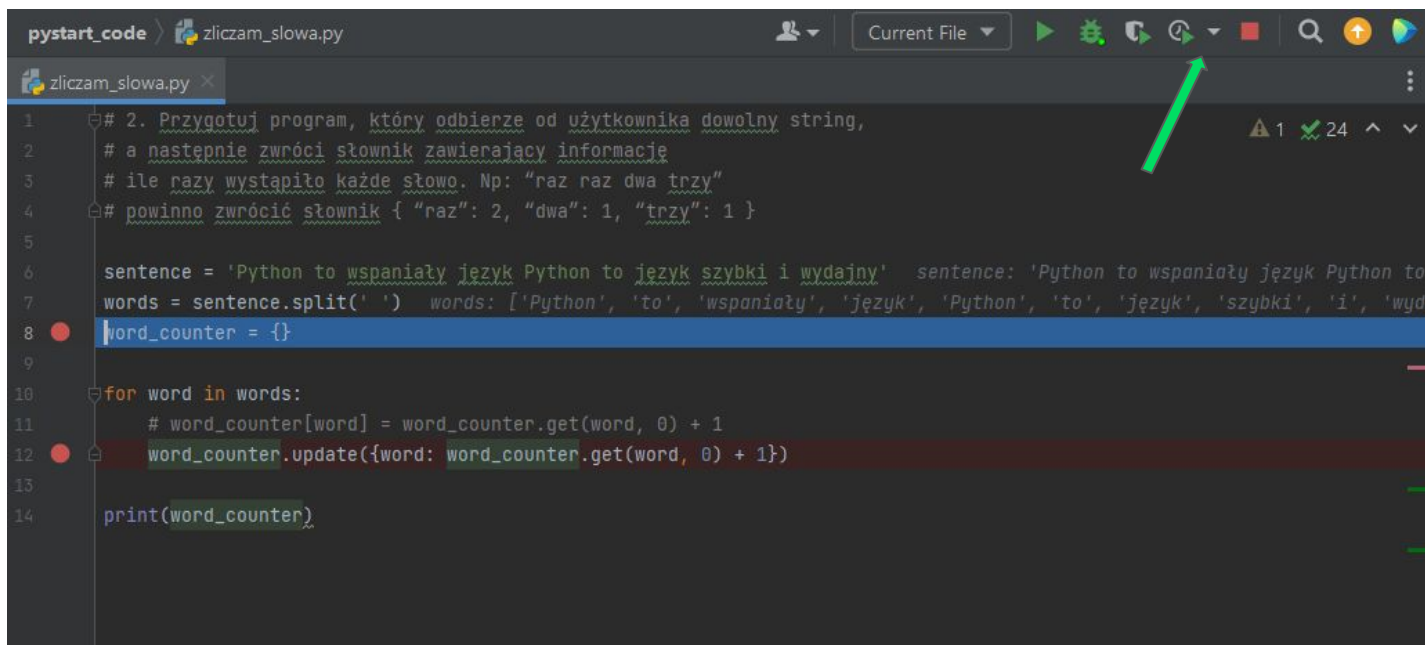
The screenshot shows a code editor with a file named `zliczam_slowa.py`. The code is a Python script for counting words in a sentence. Two breakpoints are set, indicated by red circles in the left margin with green arrows pointing to them from the left. The first breakpoint is on line 8, at the assignment `word_counter = {}`. The second breakpoint is on line 12, at the `word_counter.update` call. The code includes comments in Polish explaining the task: to take a string from the user and return a dictionary of word counts. The sentence being processed is "Python to wspaniały język Python to język szybki i wydajny".

```
1  # 2. Przygotuj program, który odbierze od użytkownika dowolny string,  
2  # a następnie zwróci słownik zawierający informacje  
3  # ile razy wystąpiło każde słowo. Np: "raz raz dwa trzy"  
4  # powinno zwrócić słownik { "raz": 2, "dwa": 1, "trzy": 1 }  
5  
6  sentence = 'Python to wspaniały język Python to język szybki i wydajny'  
7  words = sentence.split(' ')  
8  word_counter = {}  
9  
10 for word in words:  
11     # word_counter[word] = word_counter.get(word, 0) + 1  
12     word_counter.update({word: word_counter.get(word, 0) + 1})  
13  
14 print(word_counter)
```

# PyStart #18 Debugowanie aplikacji

## Dobra, jak to zrobić?

→ Po drugie włączamy debugowanie



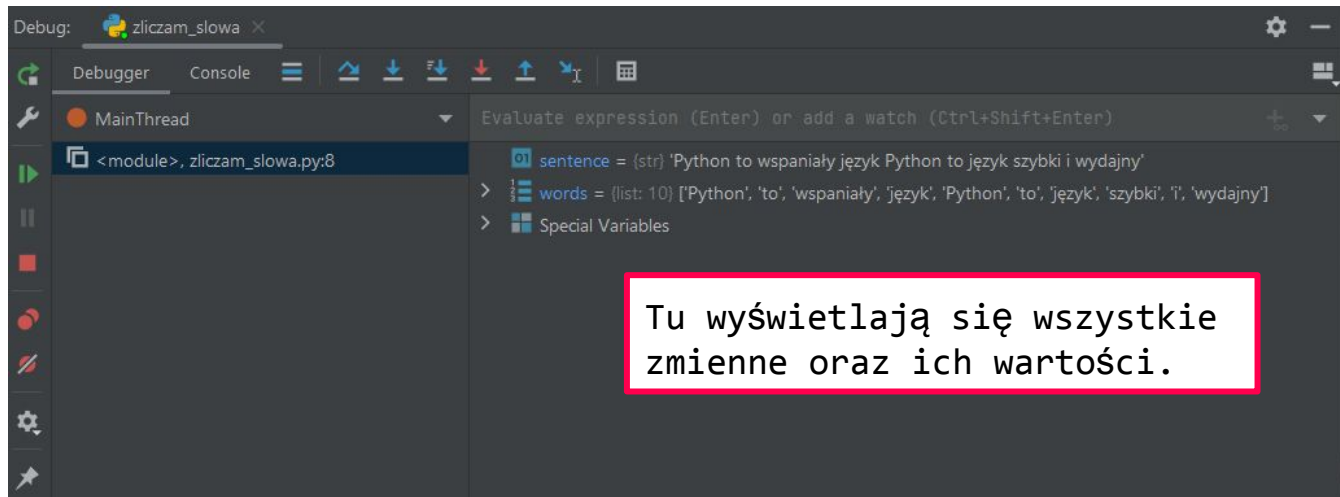
The screenshot shows a code editor with a file named `zliczam_slova.py`. The code is in Polish and implements a word counter. A green arrow points to the debug icon (a green bug) in the toolbar. The code is as follows:

```
1 # 2. Przygotuj program, który odbierze od użytkownika dowolny string,  
2 # a następnie zwróci słownik zawierający informacje  
3 # ile razy wystąpiło każde słowo. Np: "raz raz dwa trzy"  
4 # powinno zwrócić słownik { "raz": 2, "dwa": 1, "trzy": 1 }  
5  
6 sentence = 'Python to wspaniały język Python to język szybki i wydajny' sentence: 'Python to wspaniały język Python to  
7 words = sentence.split(' ') words: ['Python', 'to', 'wspaniały', 'język', 'Python', 'to', 'język', 'szybki', 'i', 'wydajny']  
8 word_counter = {}  
9  
10 for word in words:  
11     # word_counter[word] = word_counter.get(word, 0) + 1  
12     word_counter.update({word: word_counter.get(word, 0) + 1})  
13  
14 print(word_counter)
```

# PyStart #18 Debugowanie aplikacji

## Dobra, jak to zrobić?

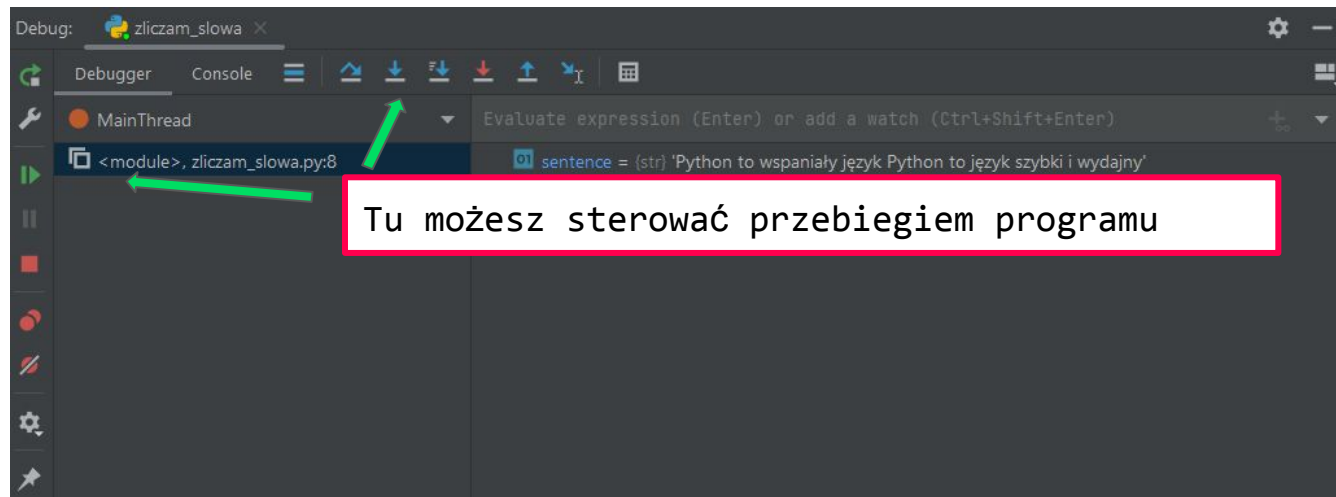
→ Po trzecie sterujemy debugowaniem



# PyStart #18 Debugowanie aplikacji

## Dobra, jak to zrobić?

→ Po trzecie sterujemy debugowaniem

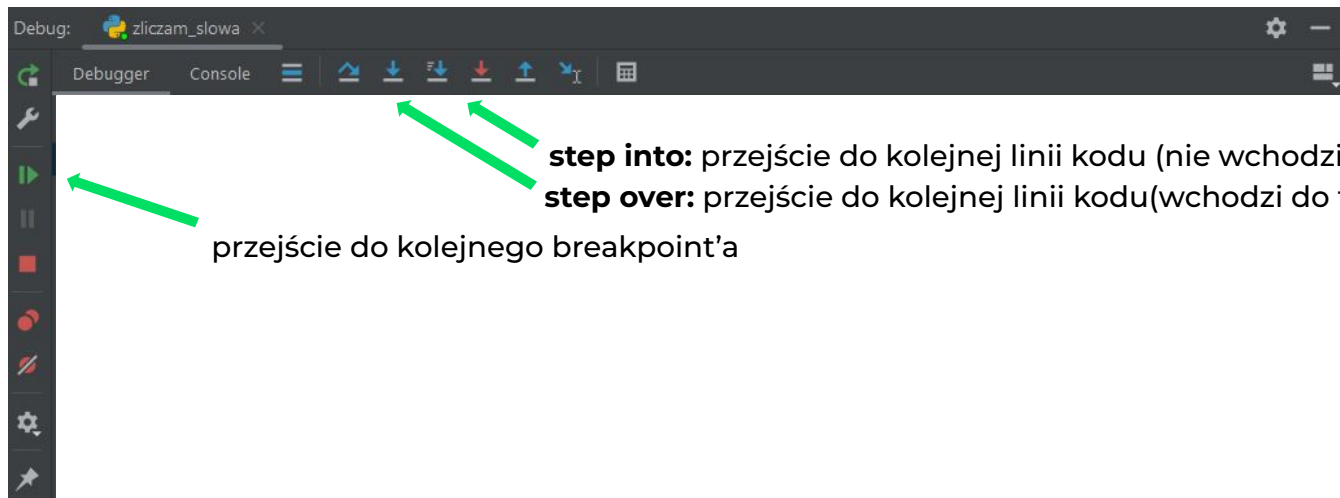




# PyStart #18 Debugowanie aplikacji

## Dobra, jak to zrobić?

→ Po trzecie sterujemy debugowaniem



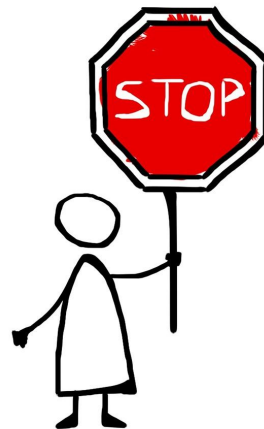
# PyStart #18 Debugowanie aplikacji

## Zadania dla nabrania wprawy

Spróbuj otworzyć w debuggerze Twój ostatni program i prześledzić kroki jego wykonywania w podobny sposób jak to zrobiliśmy wspólnie podczas demonstracji.

To zadanie nie ma rozwiązania!

Chodzi w nim o nabranie biegłości.



# PyStart #19 Match case

## Taki wygodniejszy "if"

- Pozwala na sprawdzenie kilku warunków i wykonanie kodu w zależności od tego, który warunek jest spełniony.
- Jest alternatywa dla wielu `elseif`'ów w kodzie.



```
x = 10
```

```
match x:
```

```
    case 0:
```

```
        print("x jest równe 0")
```

```
    case 10 | 20:
```

```
        print("x jest równe 10 lub 20")
```

```
    case 30:
```

```
        print("x jest równe 30")
```

```
    case _:
```

```
        print("x nie jest równe żadnej z powyższych wartości")
```

wykona się, gdy żaden inny  
case nie zadziała



# PyStart #19 Match case

## Taki wygodniejszy “if”



→ Działa również dla tupli.

```
person = ('Jan', 'Kowalski')

match person:
    case ('John', 'Doe'):
        print('Cześć John Doe!')
    case ('Jan', last_name):
        print(f'Cześć Janie o nazwisku {last_name}.')
    case (first_name, 'Kowalski'):
        print(f'Cześć kolego Kowalski o imieniu {first_name}')
    case (_, _):
        print('Nie wiem kim jesteś.')
```

# PyStart #19 Match case

## Zadania dla nabrania wprawy

Napisz program, który będzie przyjmował miesiąc w postaci liczby całkowitej od użytkownika (np. 1 dla stycznia, 2 dla lutego itd.) i wyświetlał nazwę miesiąca odpowiadającego tej liczbie. Możesz podejść do tego zadania korzystając z match'a lub użyć słownika.

```
Podaj miesiąc cyfrą od 1 do 12: 15
```

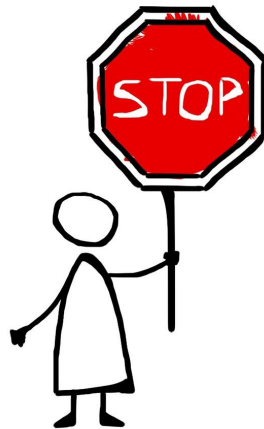
```
Nie znam takiego miesiąca!
```

```
Podaj miesiąc cyfrą od 1 do 12: 1
```

```
1 miesiąc to styczeń!
```

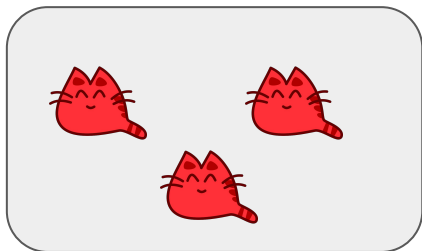
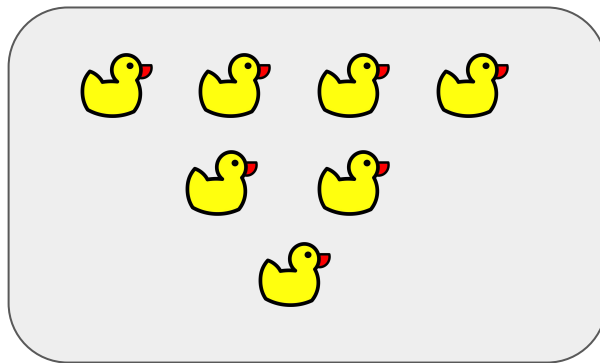
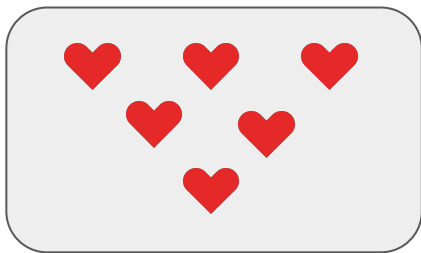
```
Podaj miesiąc cyfrą od 1 do 12: 6
```

```
6 miesiąc to czerwiec!
```



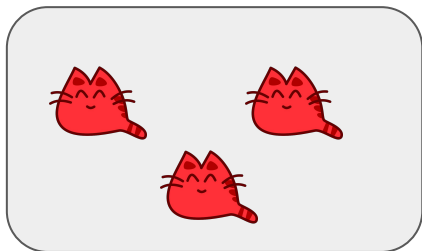
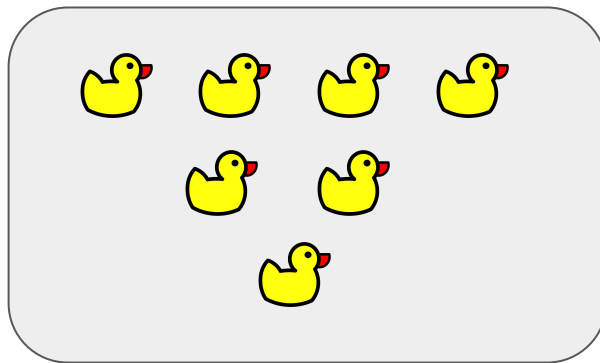
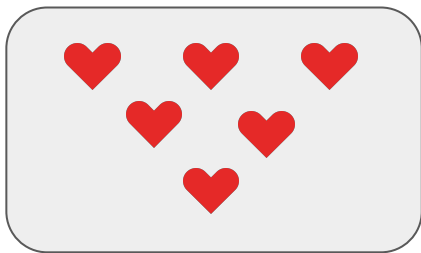
# PyStart #20 Czas na zbiory

## Co to są zbiory?



# PyStart #20 Czas na zbiory

## Co to są zbiory?



1. Przygotuj zbiór zwierząt.
2. Przygotuj zbiór czerwonych obiektów

itp...

# PyStart #20 Czas na zbiory

## Kilka informacji o zbiorach



- Są nieuporządkowane
- Możemy wykonywać na nich dodatkowe operacje
  - ◆ suma zbiorów
  - ◆ różnica zbiorów
  - ◆ część wspólna zbiorów



# PyStart #20 Czas na zbiory

## Cechy dobrego zbioru



→ Do deklaracji można użyć nawiasów klamrowych

słownik

```
>> friend = {'first_name': 'Grzegorz', 'last_name': 'Brzeczyszczykiewicz'}
```

zbiór

```
>> friends = { 'Adam', 'Kajetan', 'Kamil', 'Tomek' }
```

# PyStart #20 Czas na zbiory

## Cechy dobrego zbioru



- Do deklaracji można użyć nawiasów klamrowych
- Do rzutowania użyć funkcji `set()`

```
>> friends = ['Adam', 'Kajetan', 'Kamil', 'Tomek']  
>> set(friends)  
{'Adam', 'Kajetan', 'Kamil', 'Tomek'}
```

# PyStart #20 Czas na zbiory

## Cechy dobrego zbioru



- Do deklaracji można użyć nawiasów klamrowych
- Do rzutowania użyć funkcji `set()`
- Kolejność jest nie zachowana (!)

```
>> friends = {'Adam', 'Kajetan', 'Kamil', 'Tomek'}  
>> set(friends)  
{'Adam', 'Kajetan', 'Kamil', 'Tomek'}  
>> set(friends)  
{'Kajetan', 'Adam', 'Tomek', 'Kamil'}
```

# PyStart #20 Czas na zbiory

## Cechy dobrego zbioru



- Do deklaracji można użyć nawiasów klamrowych
- Do rzutowania użyć funkcji `set()`
- Kolejność jest nie zachowana (!)
- Nowe elementy dodajemy funkcją `.add()`

```
>> friends = {'Adam', 'Kajetan', 'Kamil', 'Tomek'}  
>> friends.add('Michał')  
{'Adam', 'Kajetan', 'Michał', 'Kamil', 'Tomek'}
```

# PyStart #20 Czas na zbiory

## Cechy dobrego zbioru

- Do deklaracji można użyć nawiasów klamrowych
- Do rzutowania użyć funkcji `set()`
- Kolejność jest nie zachowana (!)
- Nowe elementy dodajemy funkcją `.add()`
- Przechowuje tylko wartości unikatowe

```
>> bands = {'Metallica', 'Nirvana'}
```

```
>> bands.add('Nirvana')
```

```
>> bands.add('Nirvana')
```

```
>> bands
```

```
{'Metallica', 'Nirvana'}
```



# PyStart #20 Czas na zbiory

## Cechy dobrego zbioru

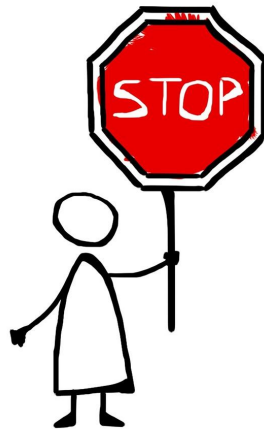


- Do deklaracji można użyć nawiasów klamrowych
- Do rzutowania użyć funkcji `set()`
- Kolejność jest nie zachowana (!)
- Nowe elementy dodajemy funkcją `.add()`
- Przechowuje tylko wartości unikatowe
- Pozwala na używanie dodatkowych operatorów na zbiorach

# PyStart #20 Czas na zbiory

## Zadania dla nabrania wprawy

1. W grze komputerowej gracz może zdobywać różne przedmioty, które są przechowywane w zbiorze. Napisz funkcję, która sprawdzi, czy gracz posiada określony przedmiot. Co stanie się, gdy drugi raz dodasz ten sam produkt?
2. Odbierz od użytkownika 10 adresów email. Sprawdź, czy adres zawiera @ oraz .com lub .pl, jeśli tak to dodaj adres do **listy**. Na koniec zamień listę na zbiór aby usunąć wszystkie duplikaty. Wyświetl ilość wpisów przed usunięciem duplikatów oraz po.



# PyStart #21 Operacje na zbiorach

## Operatory



→ suma  $|$

→ różnica  $-$

→ iloczyn  $&$

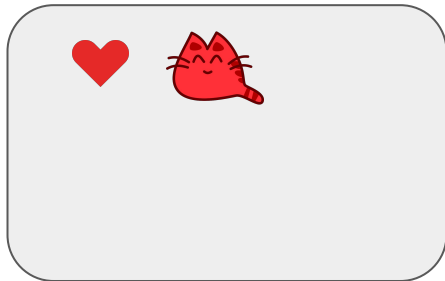
→ różnica symetryczna  $\wedge$



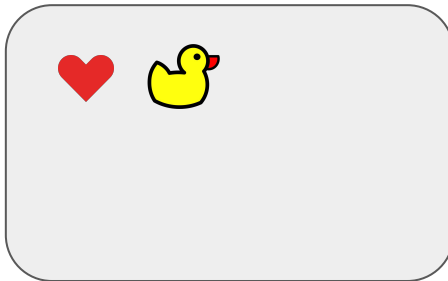
# PyStart #21 Operacje na zbiorach

## Suma

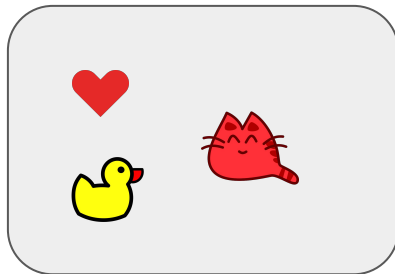
A



B



A | B

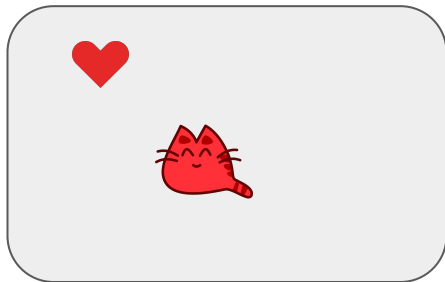


# PyStart #21 Operacje na zbiorach

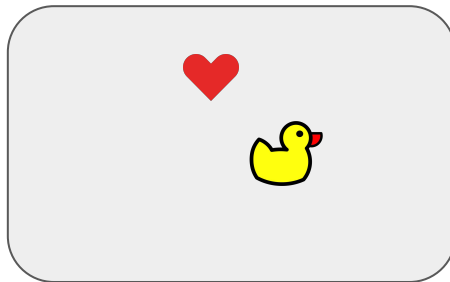
## Różnica



A



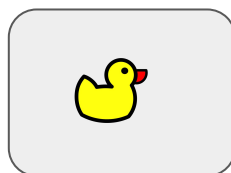
B



**A - B**



**B - A**

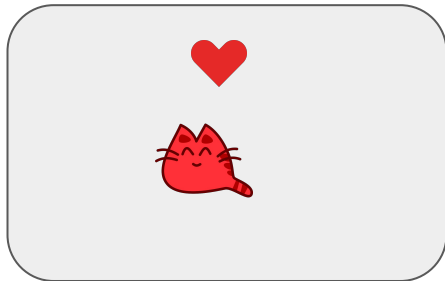


# PyStart #21 Operacje na zbiorach

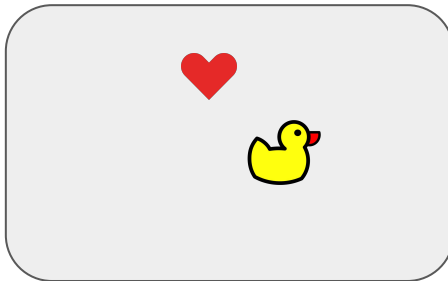
## Iloczyn



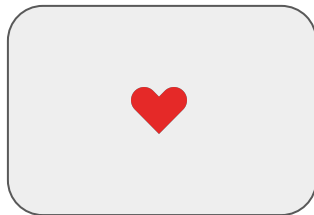
A



B



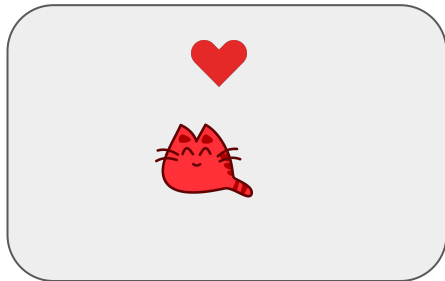
**A & B**



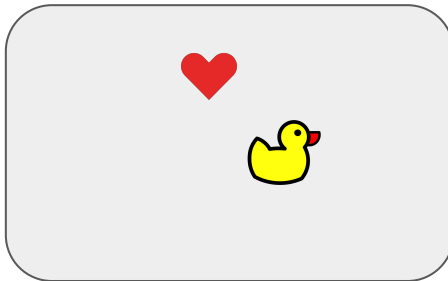
# PyStart #20 Czas na zbiory

## Różnica symetryczna

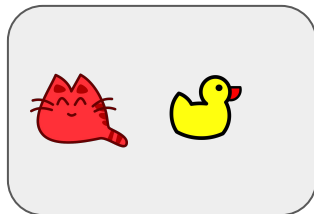
A



B



$A \Delta B$



# PyStart #21 Operacje na zbiorach

## Zadania dla nabrania wprawy

1. Przygotuj zbiór liczb podzielnych przez 3 oraz zbiór liczb podzielnych przez 5, znajdź część wspólną obu zbiorów.

2. Na podstawie dwóch słów przygotuj zbiór znaków, które nie są dla nich wspólne.

3. Masz zbiór uczniów chodzących

wieczko. Przygotuj zbiór uczniów

```
students = {  
    ("Piotr", "Kowalczyk"),  
    ("Katarzyna", "Mazur"),  
    ("Tomasz", "Adamski"),  
    ("Agnieszka", "Kaczmarek"),  
    ("Krzysztof", "Krawczyk")  
}
```

```
going_on_trip = {  
    ("Katarzyna", "Mazur"),  
    ("Tomasz", "Adamski"),  
    ("Krzysztof", "Krawczyk")  
}
```



# PyStart #22 Dopóki nie poznasz pętli while

## While? przecież już znam fora

- while czy for
- Co tak naprawdę oznacza “dopóki”?
- While true i nieskończoność



# PyStart #22 Dopóki nie poznasz pętli while

## Ogólna konstrukcja pętli while

```
while warunek:  
    instrukcja  
    instrukcja  
    instrukcja
```

Pętla będzie wykonywała się tak długo  
jak długo warunek będzie **prawdziwy**.



# PyStart #22 Dopóki nie poznasz pętli while

## Ogólna konstrukcja pętli while

```
counter = 0
while counter <= 10:
    print(counter)
    counter += 1

# wyświetli wartości od 0 do 10
```





# PyStart #22 Dopóki nie poznasz pętli while

## A co jeśli...

```
counter = 0
while counter <= 10:
    print(counter)
    # counter += 1    ...TU BĘDZIE KOMENTARZ
```

Program będzie się wykonywał w nieskończoność.  
Możemy go zatrzymać skrótem klawiszowym

**CTRL + C**



# PyStart #22 Dopóki nie poznasz pętli while

## Ciekawostka

Gdzie przydaje się pętla **while** chodząca w nieskończoność?

- Programy, które mają chodzić w tle i np. czekają na wciśnięte klawisze, programy okienkowe, skrypty które nasłuchują na żądania przesyłane z innego miejsca np. wykonujące czynność w interwale czasowym.
- Programy, które chcemy samodzielnie przerywać w inny sposób. Będziemy o tym jeszcze mówili w tej lekcji.



# PyStart #22 Dopóki nie poznasz pętli while

## Bardziej realny przykład



```
1 bus_capacity = 100
2 passengers_in_bus = 0
3
4 while bus_capacity >= passengers_in_bus:
5     passengers_in_bus += int(input('Ile osób weszło tym razem?'))
6
7     if passengers_in_bus > bus_capacity:
8         print(f'Ostatnie {passengers_in_bus-bus_capacity} musi wyjść, bo nie pojedziemy.')
9
10 print(f'Autobus rusza')
11 |
```

# PyStart #22 Dopóki nie poznasz pętli while

## Ważne słówka

- `continue` - przejście do kolejnej iteracji
- `break` - kończy pętlę w tym momencie



# PyStart #22 Dopóki nie poznasz pętli while

## Ważne słówka

- `continue` - przejście do kolejnej iteracji
- `break` - kończy pętlę w tym momencie

```
counter = 0
while True:
    print(counter)
    counter += 1
    if counter == 10:
        break
```

# wyświetli wartości od 0 do 10



# PyStart #22 Dopóki nie poznasz pętli while

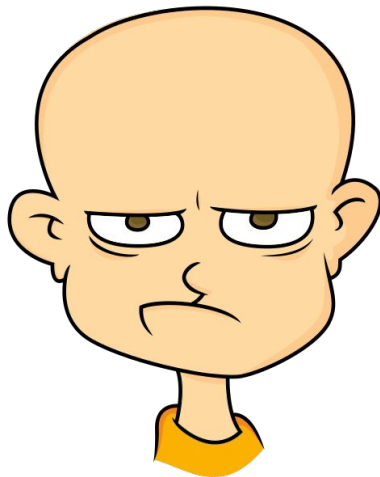
## Szczególny przypadek while



```
1 bus_capacity = 100
2 passengers_in_bus = 0
3
4 while True:
5     passengers_in_bus += int(input('Ile osób weszło tym razem?'))
6
7     should_start = input('Czy autobus ma już ruszyć? [yes/no]')
8     if should_start == 'yes':
9         break
10
11     if passengers_in_bus > bus_capacity:
12         print(f'Ostatnie {passengers_in_bus-bus_capacity} musi wyjść, bo nie pojedziemy.')
13
14 print(f'Autobus rusza')
15
```

# PyStart #22 Dopóki nie poznasz pętli while

Jakiś problem?



I co? Nikt nie musiał wysiąść?

# PyStart #22 Dopóki nie poznasz pętli while

## Szczególny przypadek while



```
1 bus_capacity = 100
2 passengers_in_bus = 0
3
4 while True:
5     passengers_in_bus += int(input('Ile osób weszło tym razem?'))
6
7     should_start = input('Czy autobus ma już ruszyć? [yes/no]')
8     if should_start == 'yes':
9         break
10
11     if passengers_in_bus > bus_capacity:
12         print(f'Ostatnie {passengers_in_bus-bus_capacity} musi wyjść, bo nie pojedziemy.')
13
14 print(f'Autobus rusza')
```



# PyStart #22 Dopóki nie poznasz pętli while

## Szczególny przypadek while

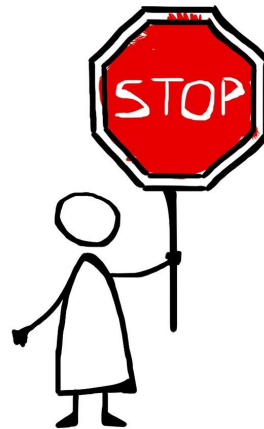


```
1 bus_capacity = 100
2 passengers_in_bus = 0
3
4 bus_can_start = False
5 while not bus_can_start:
6     passengers_in_bus += int(input('Ile osób weszło tym razem?'))
7
8     should_start = input('Czy autobus ma już ruszyć? [yes/no]')
9     if should_start == 'yes':
10         bus_can_start = True
11
12     if passengers_in_bus > bus_capacity:
13         print(f'Ostatnie {passengers_in_bus-bus_capacity} musi wyjść, bo nie pojedziemy.')
14
15 print(f'Autobus rusza')
```

# PyStart #22 Dopóki nie poznasz pętli while

## Zadania dla nabrania wprawy

1. Zapytaj użytkownika o 10 liczb dodatnich. Jeśli użytkownik poda liczbę ujemną to nie powinna być ona zliczana. Wyświetl największą i najmniejszą liczbę.
2. Poproś użytkownika o podawanie liczb, gdzie każda kolejna będzie większa od poprzedniej. Jeśli ten warunek nie nastąpi. Program powinien się zakończyć. Wyświetl średnią z tych liczb.
3. Wylosuj liczbę i poproś użytkownika by ją zgadł. Jeśli poda zbyt małą napisz „za mała”, jeśli zbyt dużą to „za duża”. Program kończy się, gdy użytkownik zgadnie wylosowaną liczbę i wyświetla ilość “strzałów” użytkownika.



# PyStart #23 Walrus operator

## Mors := (Python 3.9)

Wersja bez walrus operatora.

```
>>> a=3
>>> a
3
```

Wersja z walrus operatorem.

(W jednej instrukcji przypisuję wartość oraz zwracam)

```
>>> (a:=3)
3
>>> a
3
```



# PyStart #23 Walrus operator

## Mors :=

```
i = 0
while i < 10:
    print(i)
    i += 1
```

**Wersja bez walrus operatora.**

```
i = 0
while (i:=i+1) < 10:
    print(i)
```

**Wersja z walrus operatorem.**

(W jednej instrukcji inkrementacja oraz przypisanie)



# PyStart #23 Walrus operator

## Nie tylko pętla while!

```
n = int(input())
if n % 2 == 0:
    print(f'Liczba {n} jest parzysta')
```

```
if (n:=int(input())) % 2 == 0:
    print(f'Liczba {n} jest parzysta')
```

**Wersja bez walrus operatora.**

**Wersja z walrus operatorem.**

(W jednej instrukcji inkrementacja oraz przypisanie)

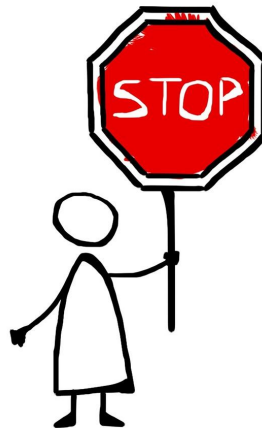


# PyStart #23 Walrus operator

## Zadania dla nabrania wprawy

Napisz program, który pobiera od użytkownika liczby całkowite i wyświetla ich sumę. Pętla while powinna działać tak długo, jak długo użytkownik wprowadzi liczbę różną od 0. Możesz użyć **walrus operatora** w warunku pętli while, aby sprawdzić, czy użytkownik wprowadził liczbę różną od 0.

```
1
2
3
4
5
0
Suma liczb wynosi: 15
```



# PyStart #24 Jak krócej zapisać if'a ?

## Skrócony if

```
1 result = 0.5
2
3 if result >= 0.6:
4     status = 'passed'
5 else:
6     status = 'failed'
7
8 print(result, status)|
9
```



## PyStart #24 Jak krócej zapisać if'a ?

### Skrócony if



```
1 result = 0.5
2
3 if result >= 0.6:
4     status = 'passed'
5 else:
6     status = 'failed'
7
8 print(result, status)
```

```
1 result = 0.5
2 status = 'failed'
3 if result >= 0.6:
4     status = 'passed'
5     |
6 print(result, status)
7
```



## PyStart #24 Jak krócej zapisać if'a ?

### Skrócony if



```
1  
2 value = 'value_if_true' if 2 == 3 else 'value_if_false'  
3 |
```

# PyStart #24 Jak krócej zapisać if'a ?

## Skrócony if

```
1 result = 0.5
2
3 if result >= 0.6:
4     status = 'passed'
5 else:
6     status = 'failed'
7
8 print(result, status)
9
```



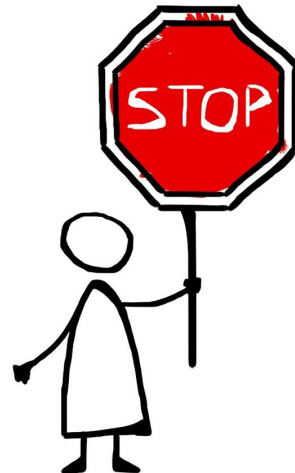
```
1 result = 0.5
2 status = 'passed' if result >= 0.6 else 'failed'
3 |
```

# PyStart #24 Skrócony if

## Zadania dla nabrania wprawy

1. Policz wynagrodzenie zatrudnionego sprzedawcy, który zarabia 2000zł za pomocą skróconej wersji if'a. Zapytaj go o staż pracy, ilość przepracowanych godzin i ilość sprzedanych towarów.

Nazwa	Wartość (od podstawy)	Warunek
Dodatek stażowy	10% pozostali 2%	powyżej 2 lat
Premia za sprzedaż	25%	powyżej 30 sztuk
Dla aktywnych	8% pozostali 2%	czas pracy 160h oraz staż powyżej 1 roku



## PyStart #25 Do czego przydaje się funkcja zip?

### Cóż to i do czego to służy?

```
1 list1 = [5, 4, 1, 2, 3, 0]
2 list2 = [6, 7, 2, 1, 3, 5]
3
4 print(list(zip(list1, list2)))
5
```

```
[(5, 6), (4, 7), (1, 2), (2, 1), (3, 3), (0, 5)]
```



# PyStart #25 Do czego przydaje się funkcja zip?

## Cóż to i do czego to służy?

Argumentów może być więcej niż dwa :)



## PyStart #25 Do czego przydaje się funkcja zip?

### Iterowanie po dwóch listach



```
1 first_names = ['Zofia', 'Leopold', 'Gabriela']
2 last_names = ['Nałkowska', 'Staff', 'Zapołska']
3
4 for first_name, last_name in zip(first_names, last_names):
5     print(f'Imię {first_name}, nazwisko {last_name}')
```

# PROJEKT #2 Czy to na pewno Twój PESEL?

## Sprawdzenie PESELu



Napisz program, który sprawdzi, czy podany przez użytkownika PESEL jest prawidłowy.  
Aby policzyć sumę kontrolną należy kolejne cyfry numeru PESEL mnożyć w następujący sposób:

Cyfra	1	2	3	4	5	6	7	8	9	10	11
Mnożnik	1	3	7	9	1	3	7	9	1	3	1

Kolejnym krokiem jest zsumowanie tych iloczynów. Jeśli ostatnia cyfra sumy tych liczb jest zerem to PESEL jest poprawny.

### Poprawne:

18210177915

$1 \times 1 + 3 \times 8 + 7 \times 2 + 9 \times 1 + 1 \times 1 \dots$  itd..

99010124415

# CZWARTA PRACA DOMOWA

## UWAGA! UWAGA!



Korzystając z match case:

**Zadanie 1:** Napisz program, który będzie wykonywał różne operacje matematyczne w zależności od wyboru użytkownika. Użytkownik powinien mieć możliwość wyboru między dodawaniem, odejmowaniem, mnożeniem i dzieleniem.

**Zadanie 2:** Napisz program, który będzie konwertował różne jednostki masy na podstawie wyboru użytkownika. Użytkownik powinien mieć możliwość konwersji między kilogramami, funtami i uncjami.

Przelicznik jest 1 kilogram = 35.274 uncji oraz 1 kilogram = 2.20462 funtów



# CZWARTA PRACA DOMOWA

## UWAGA! UWAGA!



### Pętla while

**Zadanie 3:** Zdefiniuj hasło w zmiennej, a następnie napisz program, który za pomocą pętli while będzie pytać użytkownika o wprowadzenie hasła. Jeśli hasło jest poprawne, program powinien wyświetlić komunikat "Poprawne hasło", a jeśli hasło jest niepoprawne, program powinien wyświetlić komunikat "Niepoprawne hasło, spróbuj ponownie".

**Zadanie 4:** Napisz program symulujący działanie bankomatu. Pętla while będzie pytać użytkownika o wybór operacji (np. wpłata, wypłata, sprawdzenie stanu konta) i będzie kontynuowana dopóki użytkownik nie wybierze opcji "wyjście".

# CZWARTA PRACA DOMOWA

## UWAGA! UWAGA!



### Pętla while

**Zadanie 5:** Napisz program, który będzie wykonywał pętlę while, która będzie pytać użytkownika o nazwę miasta oraz temperaturę w stopniach Celsjusza. Program będzie przechowywał te informacje w słowniku, gdzie kluczem będzie nazwa miasta, a wartością temperatura. Pętla będzie kontynuowana dopóki użytkownik nie wprowadzi "koniec". Po zakończeniu pętli program powinien wyświetlić średnią temperaturę dla wszystkich miast oraz nazwę miasta o najwyższej i najniższej temperaturze.

**Zadanie 6:** Pytaj użytkownika o imion dopóki nie napisze "koniec" program powinien wyświetlić słownik prezentujący imiona wraz z ilością ich wystąpień oraz unikatową listę imion posortowanych alfabetycznie.

# CZWARTA PRACA DOMOWA

## UWAGA! UWAGA!



### Pętla while

**Zadanie 7:** Napisz program, który będzie wykonywał pętlę while, która będzie pytała użytkownika o wprowadzenie liczby. Program będzie sprawdzał czy liczba jest parzysta i jeśli tak, to będzie ją dodawał do listy liczb parzystych, a jeśli nie, to do listy liczb nieparzystych. Pętla będzie kontynuowana dopóki użytkownik nie wprowadzi liczby 0. Po zakończeniu pętli program powinien wyświetlić liczbę liczb parzystych i nieparzystych oraz listy liczb parzystych i nieparzystych.

# CZWARTA PRACA DOMOWA

**UWAGA! UWAGA!**



## Zadanie 8: Sklep

Przygotuj słownik zawierający produkty. Produkt to klucz, a cena to wartość. Zapytaj użytkownika który produkt chce dodać do koszyka, a następnie w jakiej ilości. Pytaj dopóki użytkownik nie odpowie “podsumuj”.

Wartość zamówienia możesz przechowywać w pojedynczej zmiennej lub posiadać listę (dla chętnych) produktów lub słowników jeśli chcesz wyświetlić podsumowanie.

# CZWARTA PRACA DOMOWA

## UWAGA! UWAGA!



### Praca ze stringami

**Zadanie 9:** Przygotuj program, który będzie szyfrował i odszyfrowywał wyrażenia korzystając z szyfru Cezara. Podpowiedź: mogą Ci się przydać funkcję `ord()` oraz `chr()`.

**Zadanie 10:** Napisz program, który będzie usuwał z tekstu wszystkie słowa zaczynające się na wybraną literę.

**Zadanie 11:** Zapytaj użytkownika o dwa wyrażenia, a następnie wyświetl wszystkie znaki wspólne dla obu tych wyrażen. np. sala i balkon: powinno wyświetlić `a` oraz `l`

**Zadanie 12:** Odbierz od użytkownika słowa rozdzielone przecinkami, a następnie wyświetl te słowa wiersz pod wierszem, ale każde tylko jeden raz.

# CZWARTA PRACA DOMOWA

## UWAGA! UWAGA!



- Przygotuj program, który będzie sprawdzał czy dwa podane słowa są wzajemnymi anagramami

