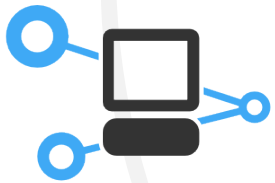
Programowanie funkcyjne w Pythonie

Koło DSG 2013

Konrad Siek

konrad.siek@cs.put.edu.pl

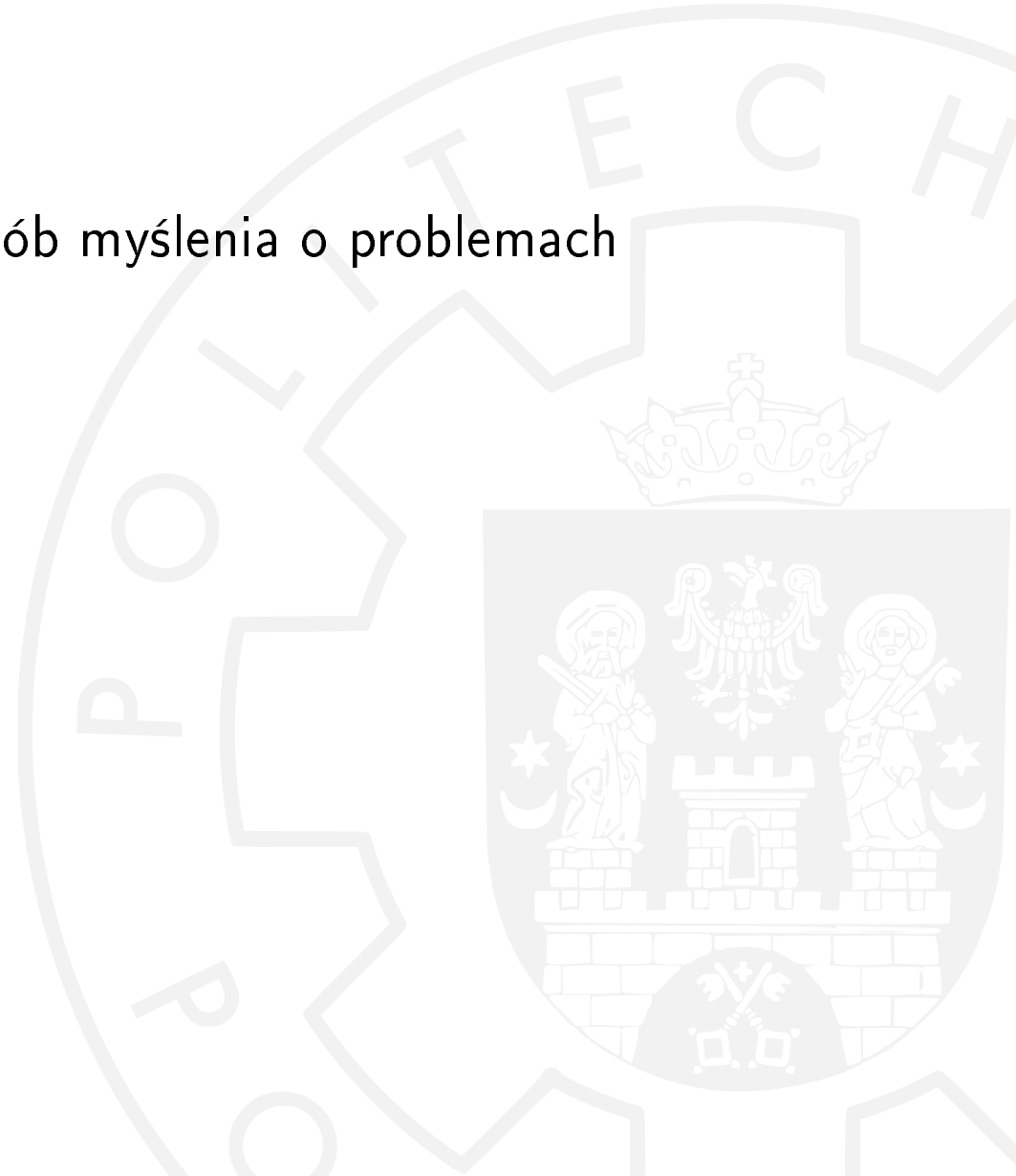


Distributed Systems Group

dsg.cs.put.poznan.pl

Paradygmat funkcyjny

Paradygmat = sposób myślenia o problemach

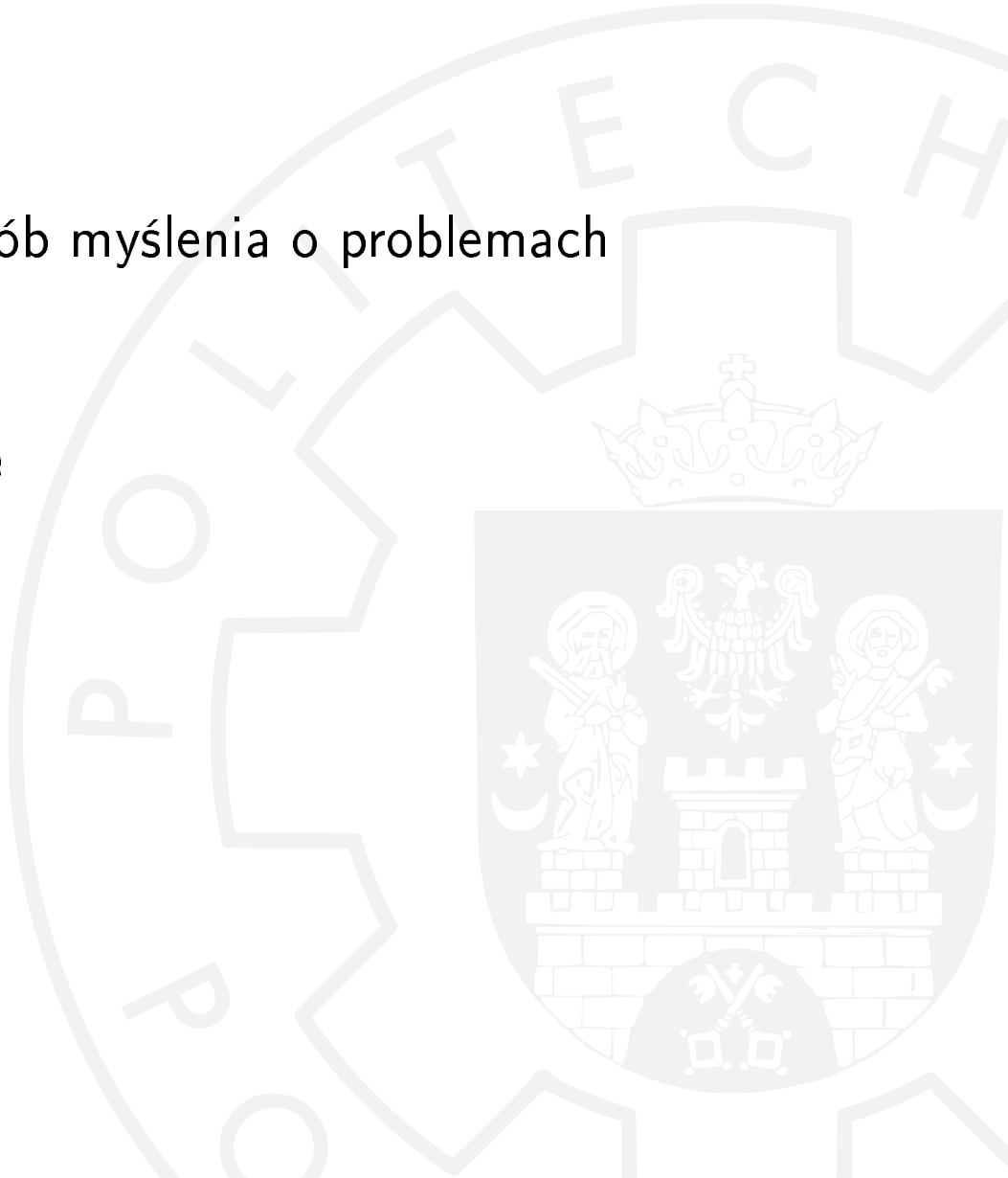


Paradygmat funkcyjny

Paradygmat = sposób myślenia o problemach

Programowanie imperatywne

- co robić?
- w jakiej kolejności?
- pętle, procedury, obiekty



Paradygmat funkcyjny

Paradygmat = sposób myślenia o problemach

Programowanie imperatywne

- co robić?
- w jakiej kolejności?
- pętle, procedury, obiekty

Programowanie funkcyjne

- jaki ma być efekt?
- jak skomponować program?
- funkcje i rekurencja

Problem

$$\left(\sum_{e \in \texttt{ls}} e \right)$$

suma elementów listy `ls`

Problem

$$\left(\sum_{e \in \texttt{ls}} e \right)$$

suma elementów listy `ls`

styl imperatywny

```
sum = 0
for e in ls
    sum += e
```

Problem

$$\left(\sum_{e \in \texttt{ls}} e \right)$$

suma elementów listy `ls`

styl imperatywny

```
sum = 0
for e in ls
    sum += e
```

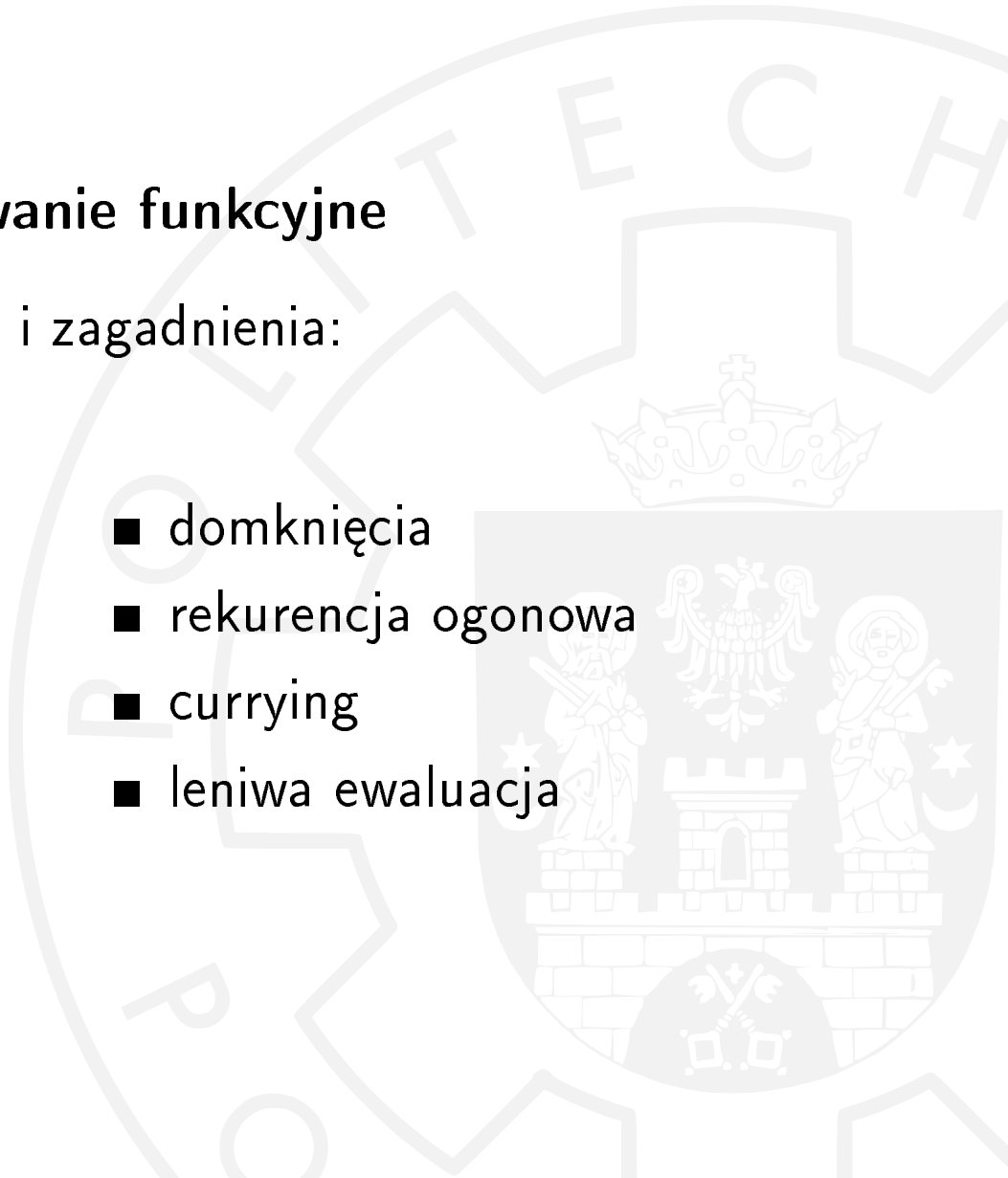
styl funkcyjny

```
def f(ls):
    if not ls:
        return 0
    else:
        return ls[0] + f(ls[1:])
```

Programowanie funkcyjne

Koncepcje i zagadnienia:

- funkcje pierwszej klasy
- funkcje wyższego rzędu
- czyste funkcje
- funkcje anonimowe
- domknięcia
- rekurencja ogonowa
- currying
- leniwa ewaluacja



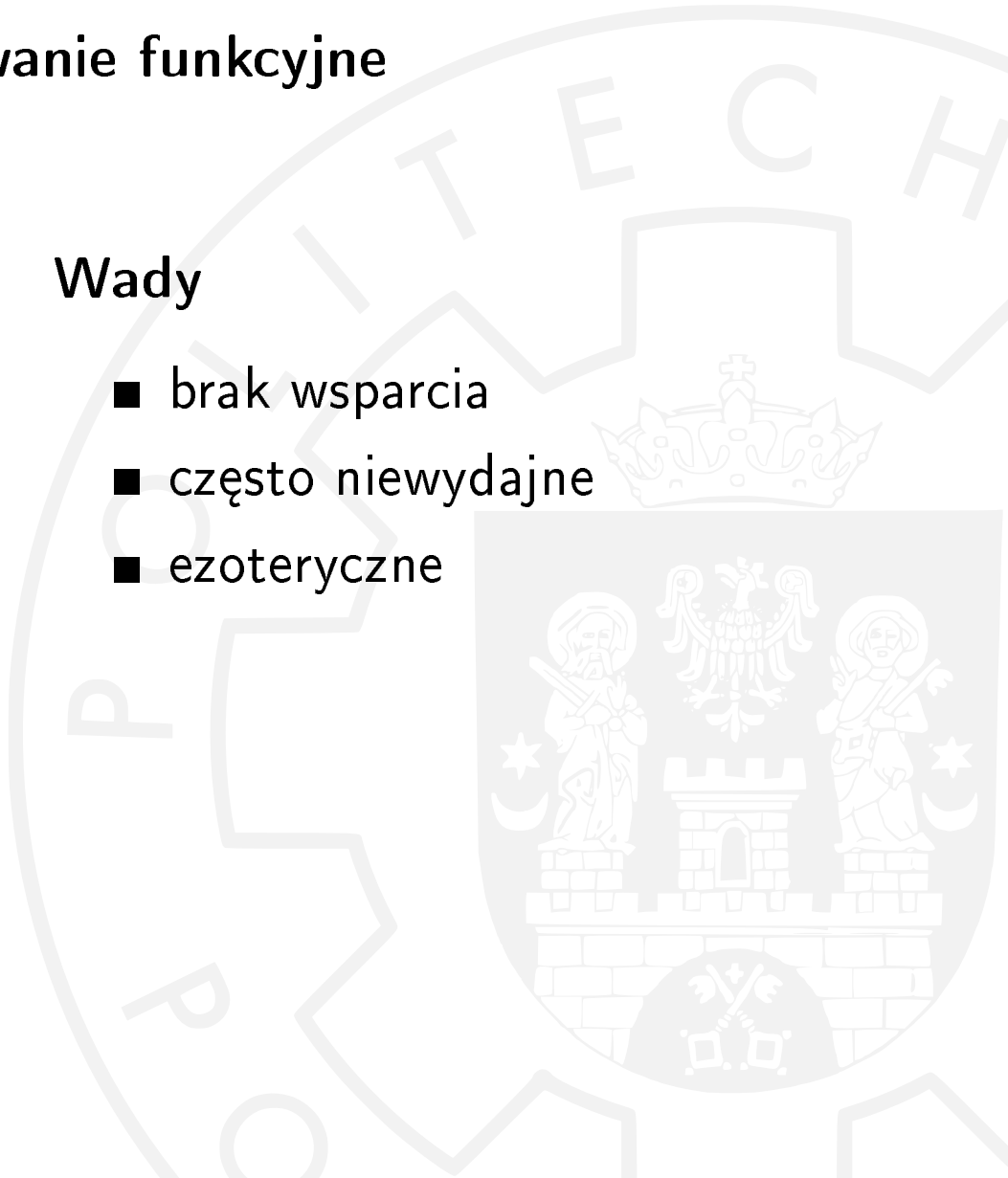
Programowanie funkcyjne

Zalety

- kompozycja
- współbieżność
- testowanie
- udawadnianie
- pasuje do niektórych problemów

Wady

- brak wsparcia
- często niewydajne
- ezoteryczne



Paradygmat funkcyjny

Programowanie funkcyjne

Zalety

- kompozycja
- współbieżność
- testowanie
- udawadnianie
- pasuje do niektórych problemów

Wady

- brak wsparcia
- często niewydajne
- ezoteryczne

Uzupełnienia inne paradygmaty

Funkcje pierwszej klasy

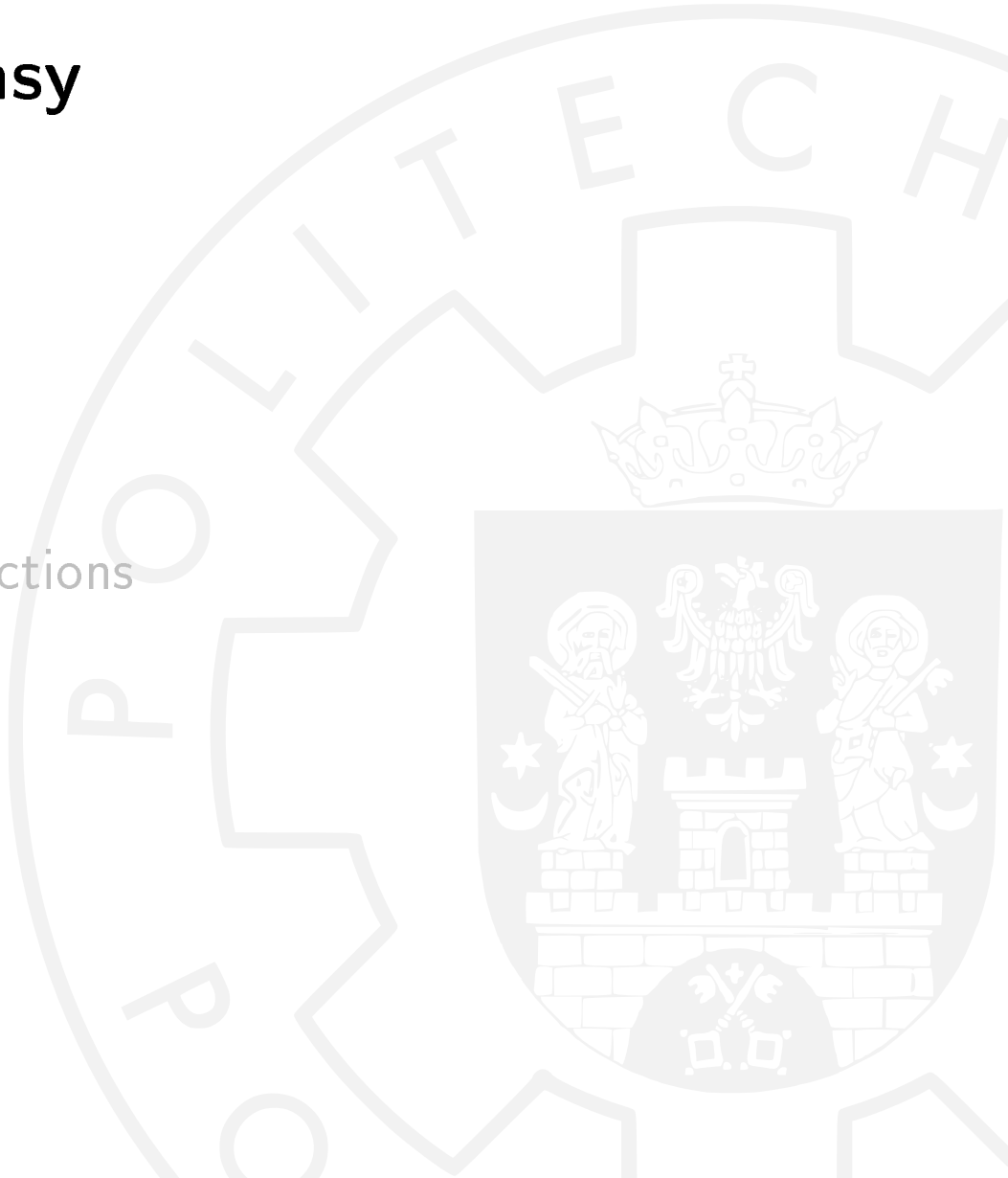
map, filter, reduce

first-class, higher order functions

lambda

underscore variable

list comprehension



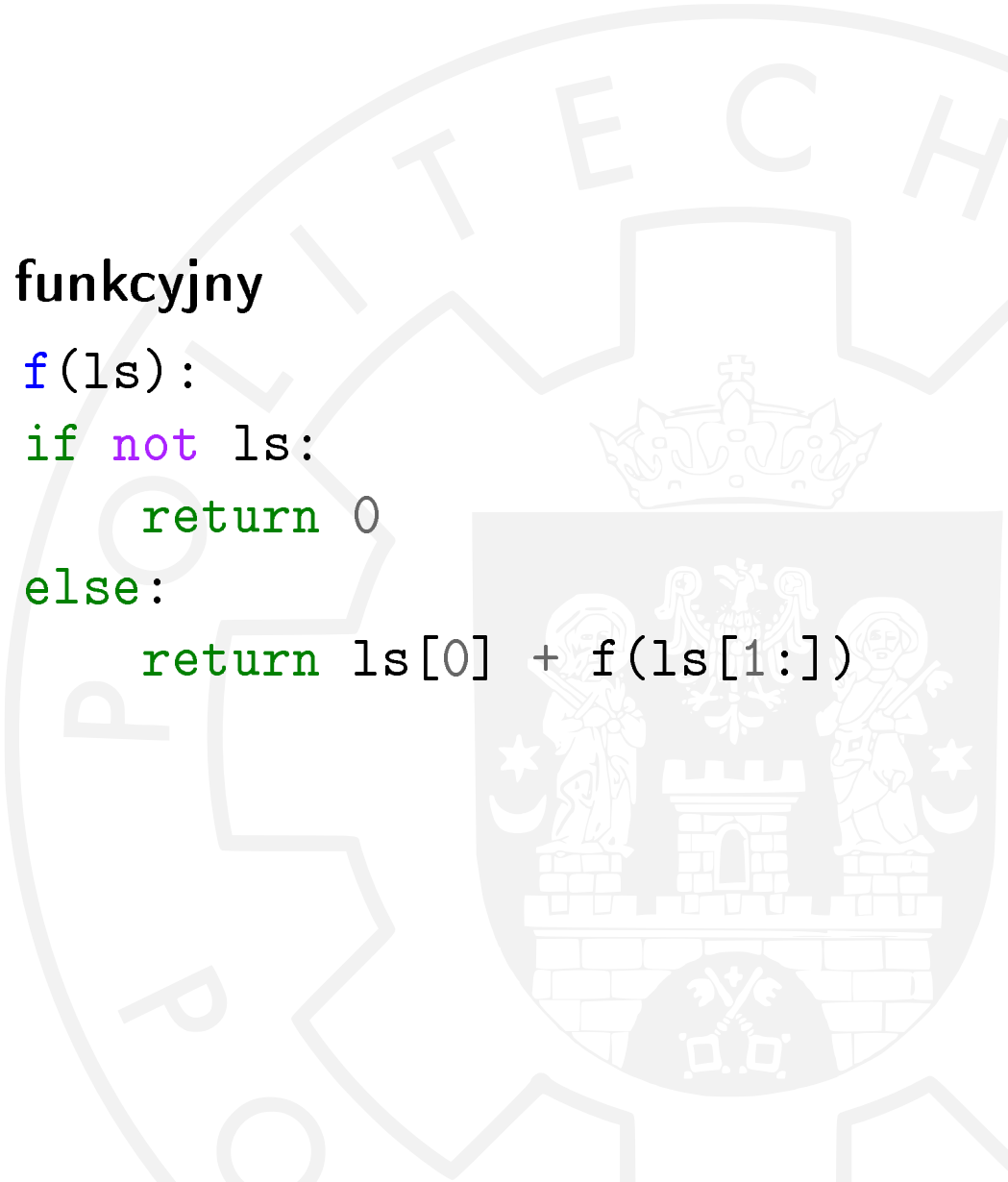
Funkcje pierwszej klasy

styl imperatywny

```
sum = 0
for e in ls
    sum += e
```

styl funkcyjny

```
def f(ls):
    if not ls:
        return 0
    else:
        return ls[0] + f(ls[1:])
```



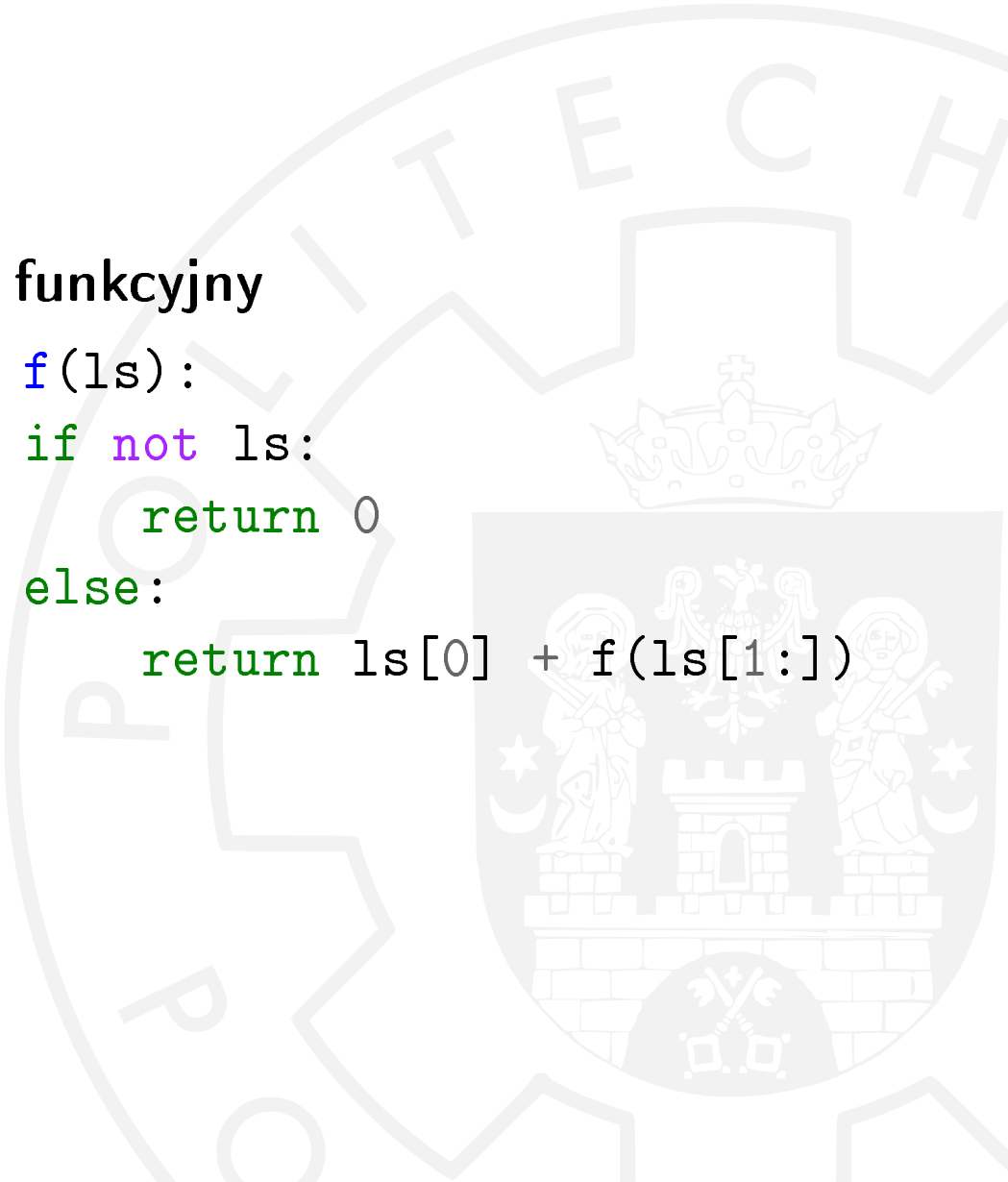
Funkcje pierwszej klasy

styl imperatywny

```
i, sum = 0, 0
while i < length(ls):
    sum += ls[i]
    i += 1
```

styl funkcyjny

```
def f(ls):
    if not ls:
        return 0
    else:
        return ls[0] + f(ls[1:])
```

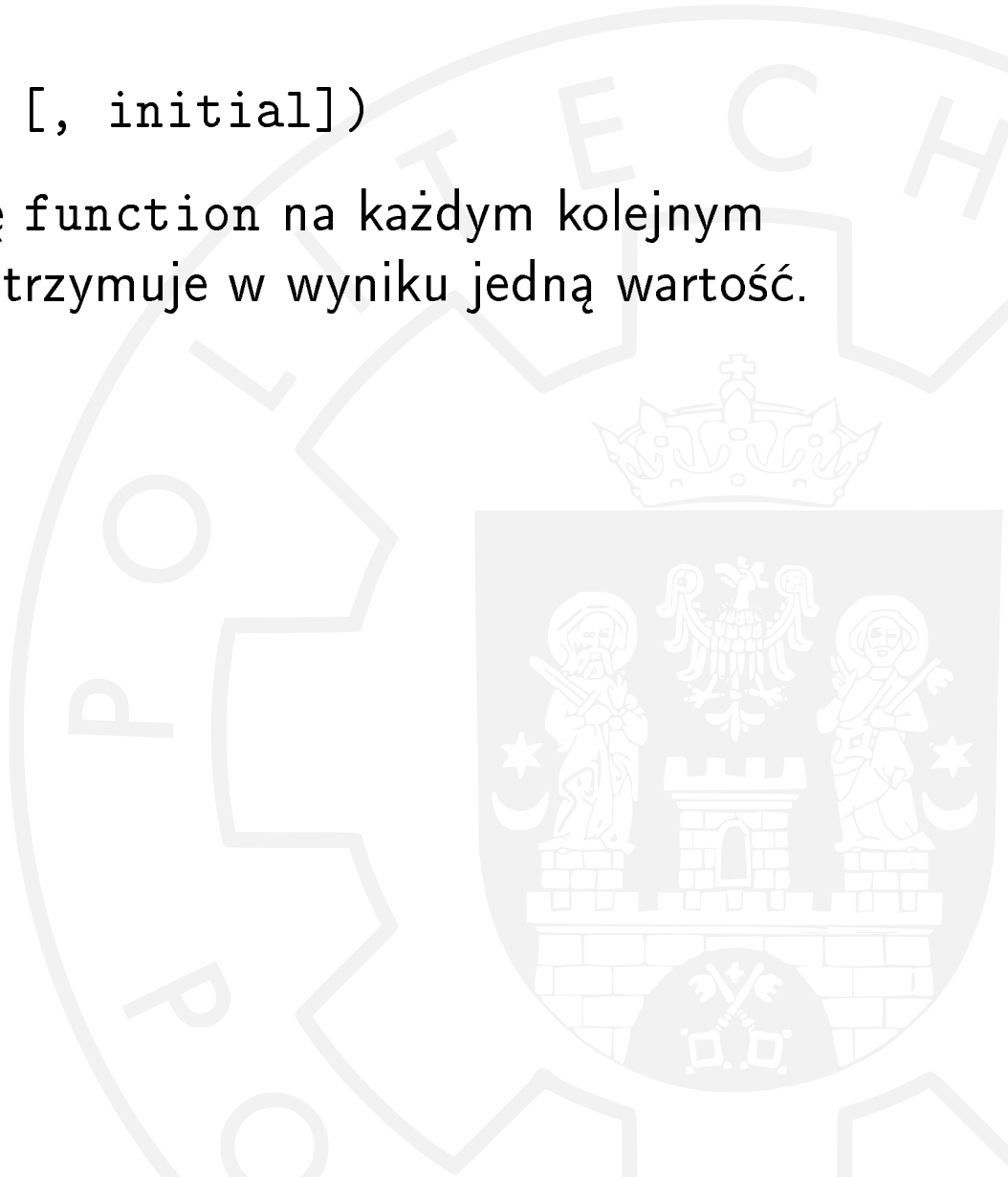


Funkcje pierwszej klasy

```
reduce(function, sequence [, initial])
```

Uruchamia **kumulującą** funkcję `function` na każdym kolejnym elemencie kolekcji `sequence` i otrzymuje w wyniku jedną wartość.

Python3: `functools.reduce`



Funkcje pierwszej klasy

```
reduce(function, sequence [, initial])
```

Uruchamia **kumulującą** funkcję `function` na każdym kolejnym elemencie kolekcji `sequence` i otrzymuje w wyniku jedną wartość.

Python3: `functools.reduce`

```
def reduce(function, sequence, initial):
    if not sequence:
        return initial
    else:
        return function(sequence[0],
            reduce(function, sequence[1:], initial)
```

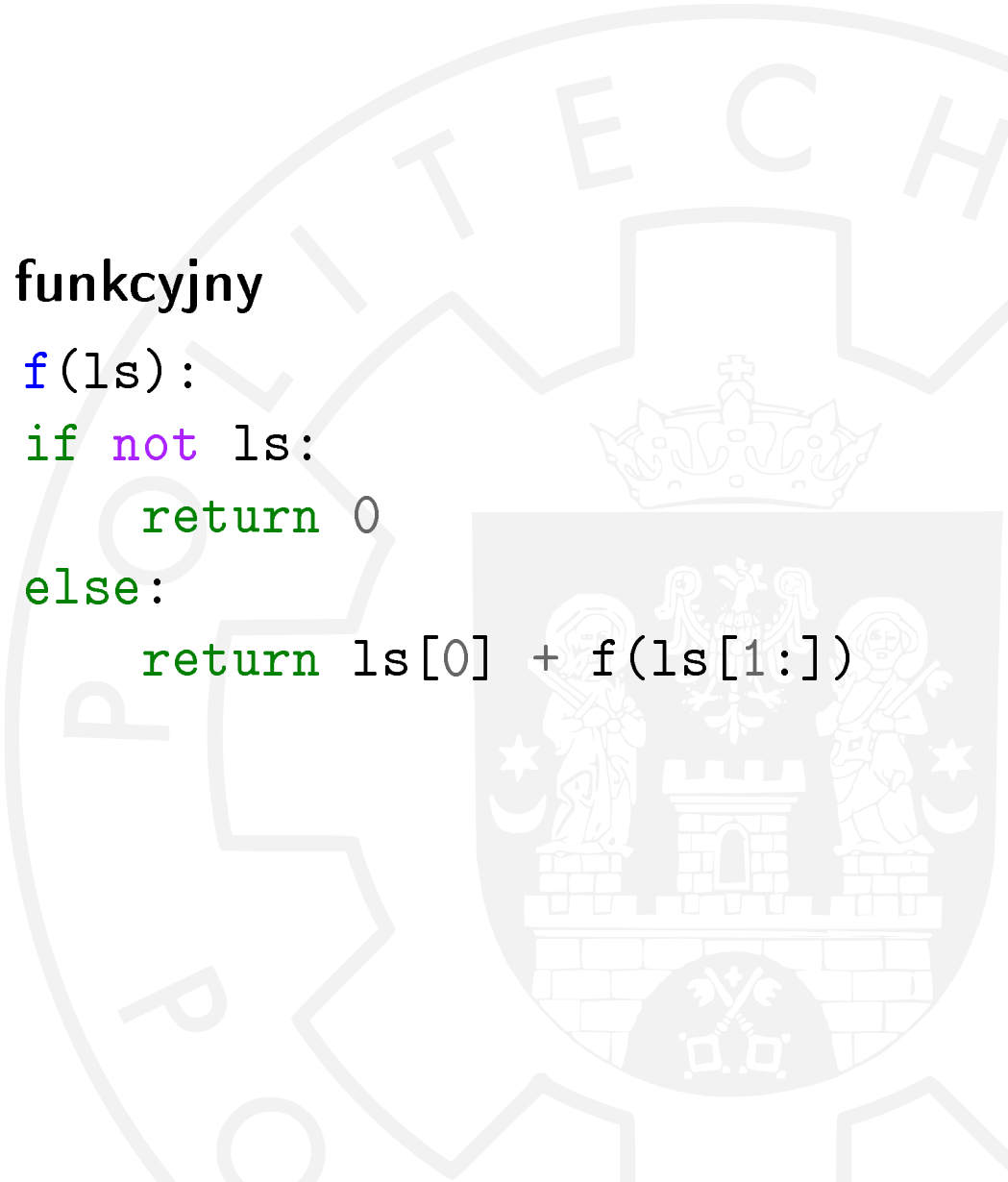
Funkcje pierwszej klasy

styl imperatywny

```
sum = 0
for e in ls
    sum += e
```

styl funkcyjny

```
def f(ls):
    if not ls:
        return 0
    else:
        return ls[0] + f(ls[1:])
```



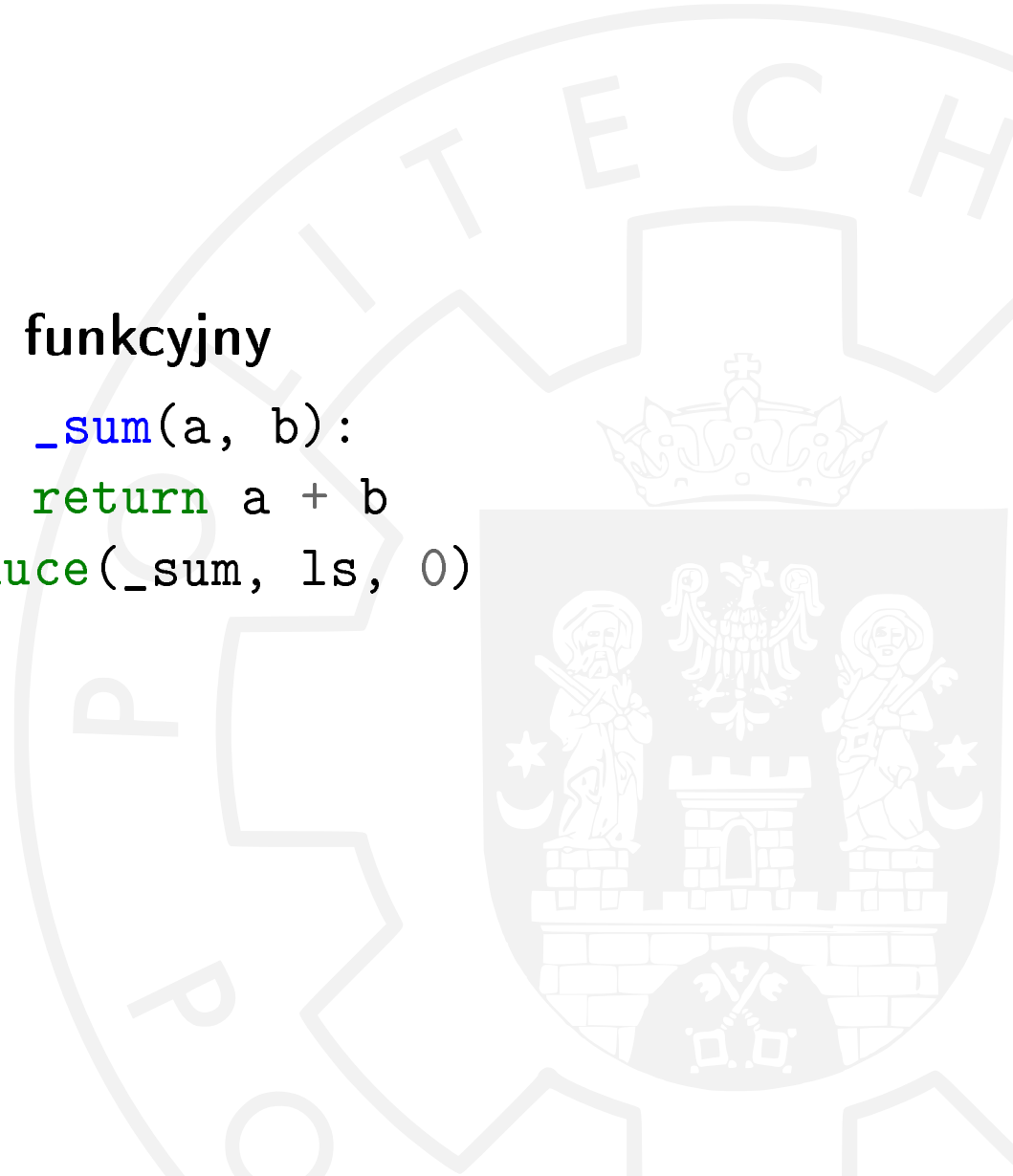
Funkcje pierwszej klasy

styl imperatywny

```
sum = 0
for e in ls
    sum += e
```

styl funkcyjny

```
def _sum(a, b):
    return a + b
reduce(_sum, ls, 0)
```



Funkcje pierwszej klasy

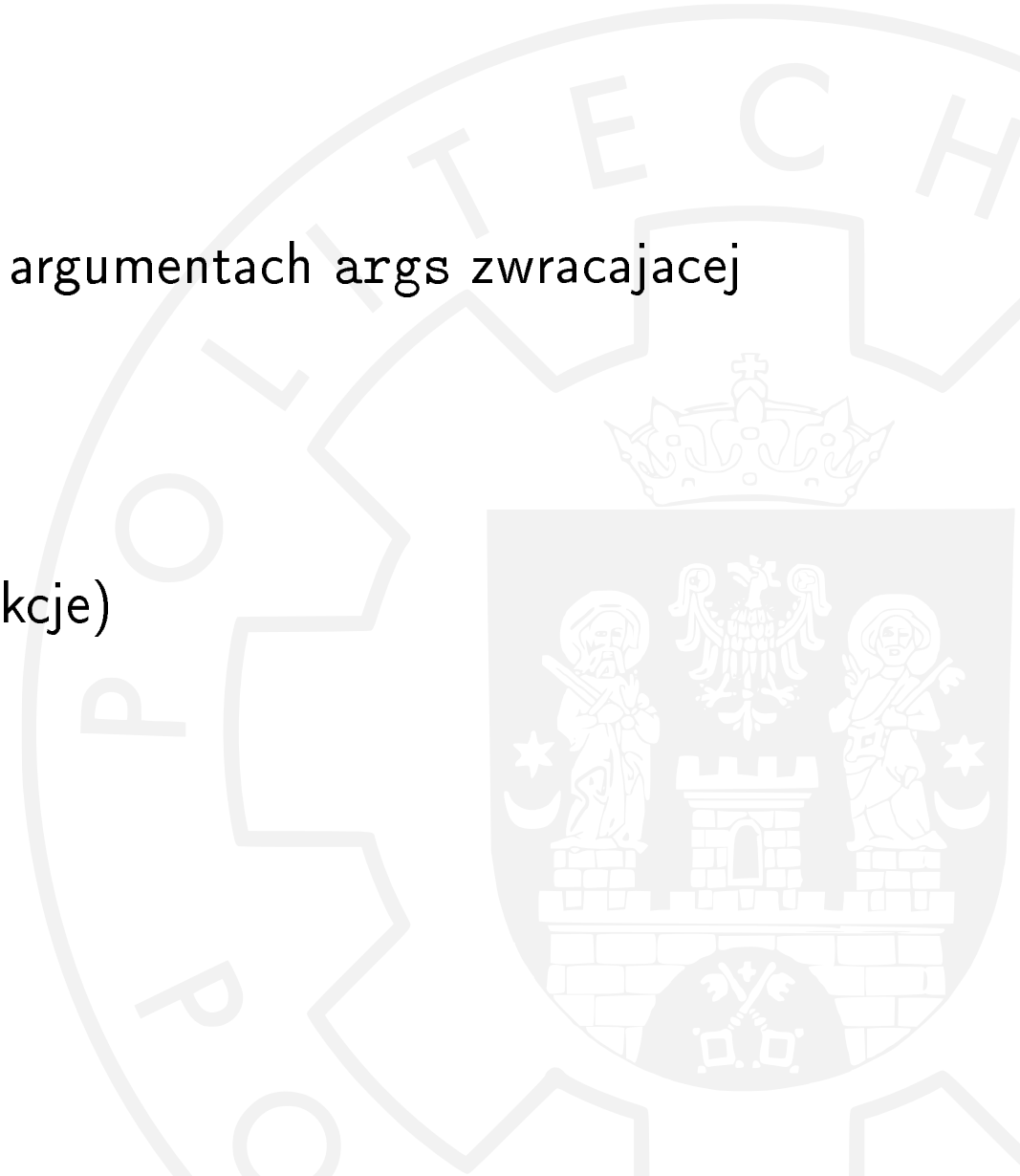
```
lambda [args]: expr
```

Operator funkcji anonimowej o argumentach `args` zwracajacej wynik wyrażenia `expr`.

Ograniczenia:

- tylko jedno wyrażenie
- tylko wyrażenia (nie instrukcje)

```
square = lambda x: x^2
square(2)
```



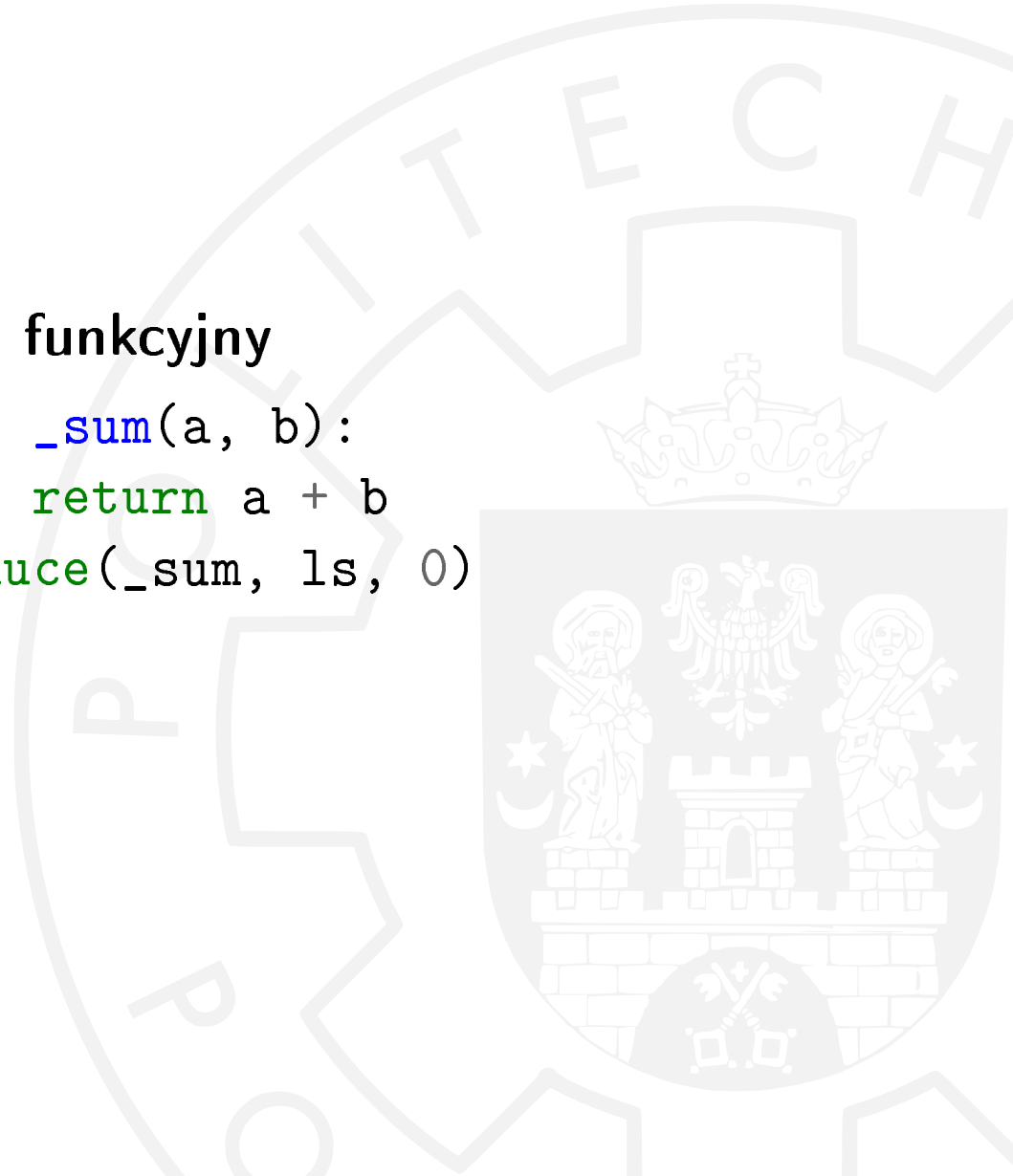
Funkcje pierwszej klasy

styl imperatywny

```
sum = 0
for e in ls
    sum += e
```

styl funkcyjny

```
def _sum(a, b):
    return a + b
reduce(_sum, ls, 0)
```



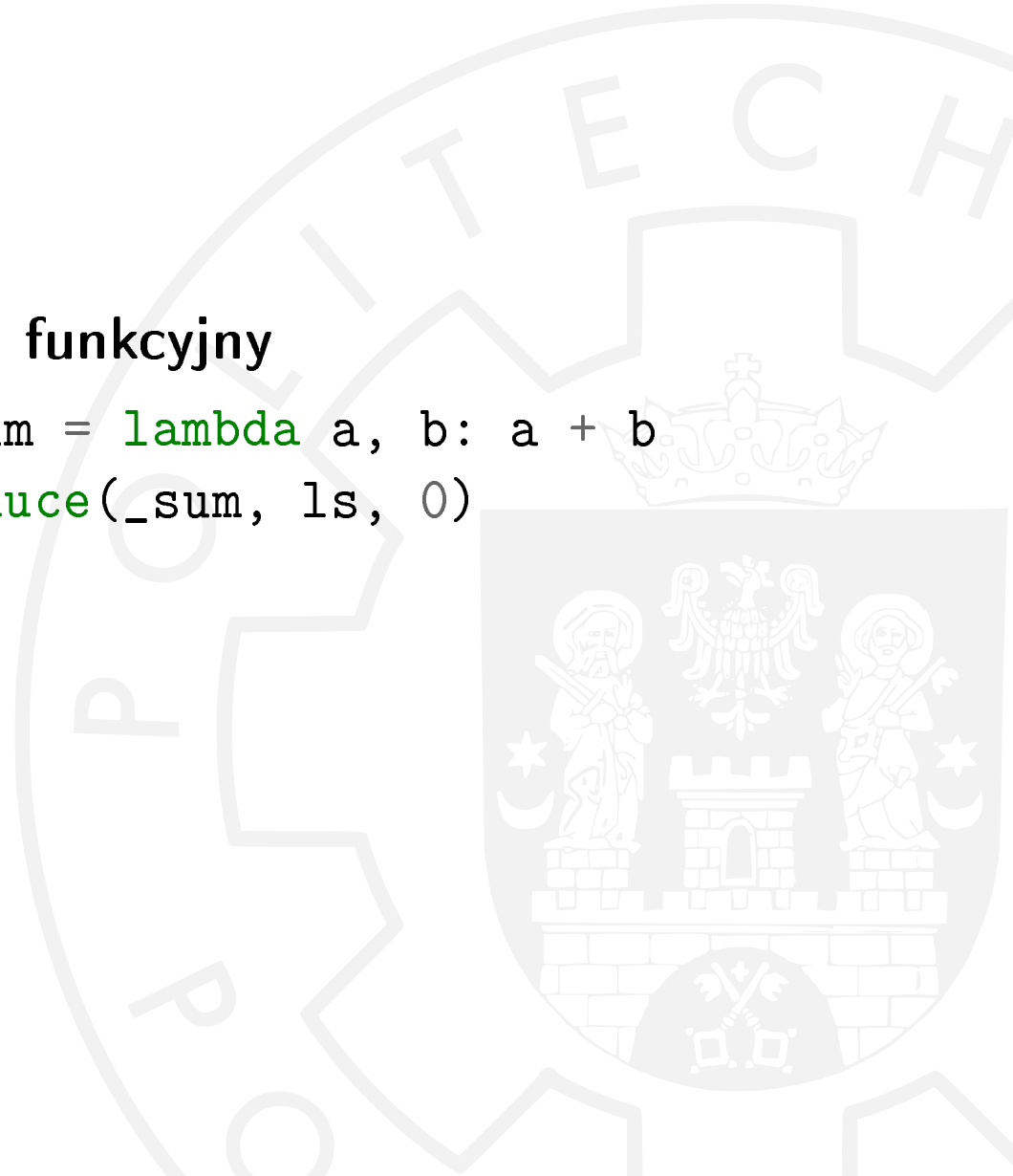
Funkcje pierwszej klasy

styl imperatywny

```
sum = 0
for e in ls
    sum += e
```

styl funkcyjny

```
_sum = lambda a, b: a + b
reduce(_sum, ls, 0)
```



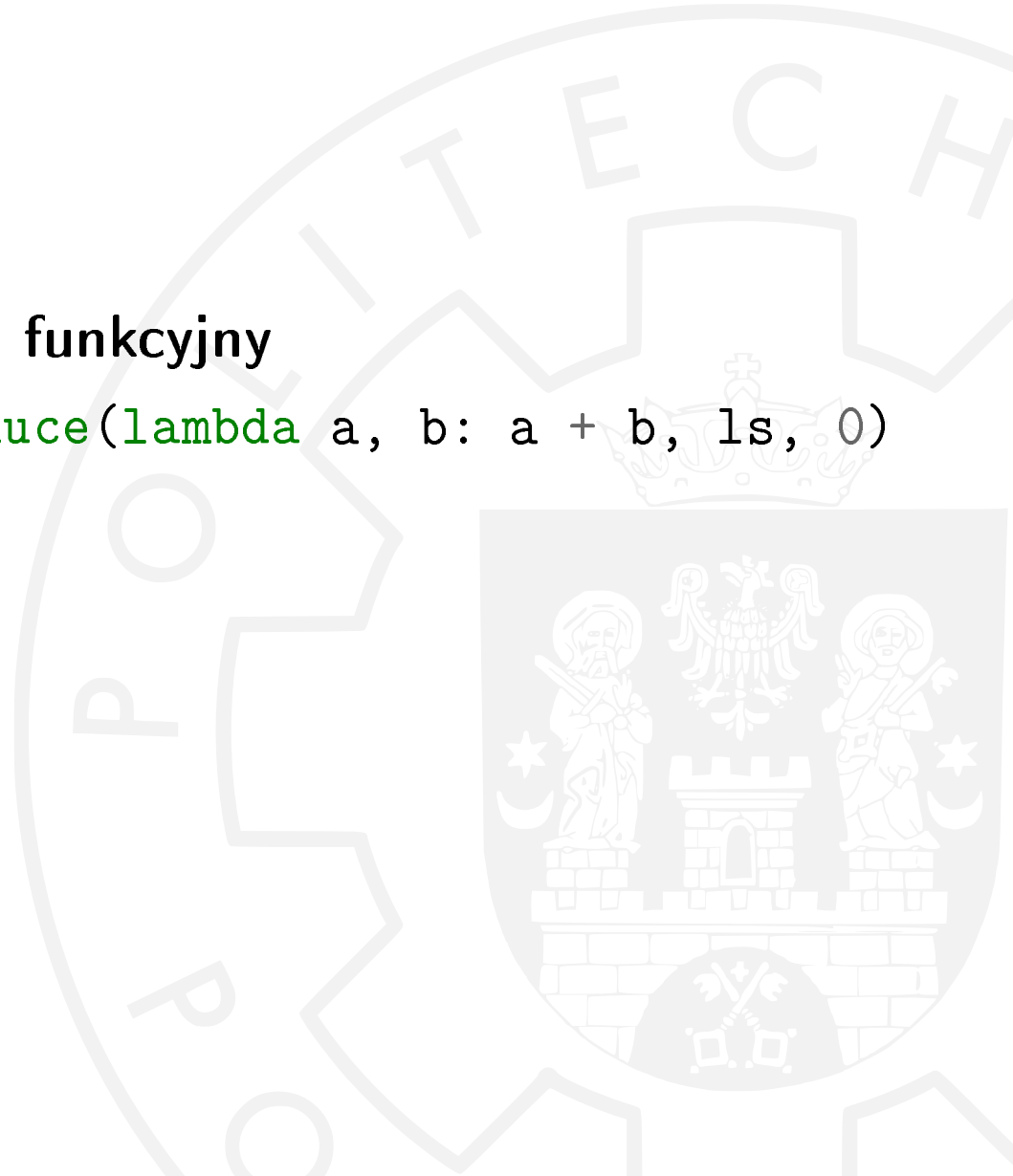
Funkcje pierwszej klasy

styl imperatywny

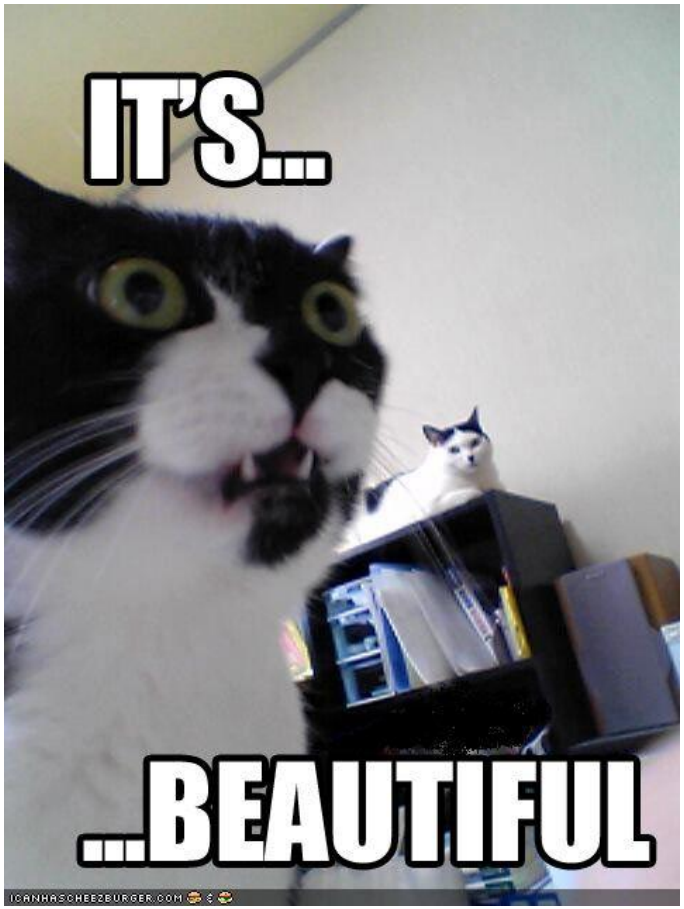
```
sum = 0
for e in ls
    sum += e
```

styl funkcyjny

```
reduce(lambda a, b: a + b, ls, 0)
```



Funkcje pierwszej klasy



styl funkcyjny

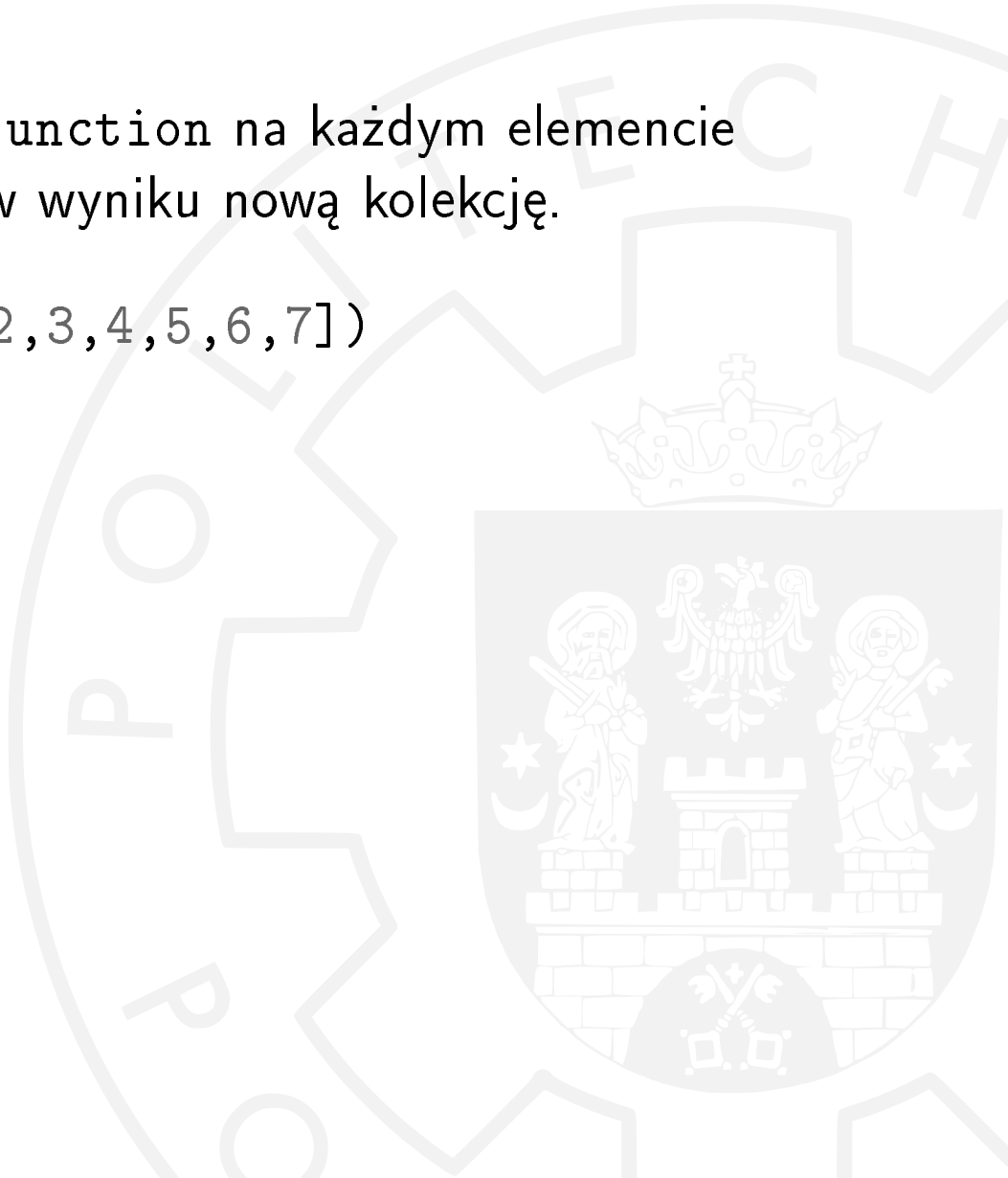
```
reduce(lambda a, b: a + b, ls, 0)
```

Funkcje pierwszej klasy

```
map(function, sequence)
```

Uruchamia **mapującą** funkcję `function` na każdym elemencie kolekcji `sequence` i otrzymuje w wyniku nową kolekcję.

```
map(lambda x: 2**x, [0,1,2,3,4,5,6,7])
```



Funkcje pierwszej klasy

```
map(function, sequence)
```

Uruchamia **mapującą** funkcję `function` na każdym elemencie kolekcji `sequence` i otrzymuje w wyniku nową kolekcję.

```
map(lambda x: 2**x, [0,1,2,3,4,5,6,7])
```

```
filter(function, sequence)
```

Uruchamia **filtrującą** funkcję `function` na każdym elemencie kolekcji `sequence` i otrzymuje w wyniku nową kolekcję zawierającą tylko te elementy dla których `function` zwróciło prawdę.

```
filter(lambda x: x and x % 2 == 0, [0,1,2,3,4,5,6,7])
```

Funkcje pierwszej klasy

```
map(function, sequence)
```

Uruchamia **mapującą** funkcję `function` na każdym elemencie kolekcji `sequence` i otrzymuje w wyniku nową kolekcję.

```
map(lambda x: 2**x, [0,1,2,3,4,5,6,7])
```

```
filter(function, sequence)
```

Uruchamia **filtrującą** funkcję `function` na każdym elemencie kolekcji `sequence` i otrzymuje w wyniku nową kolekcję zawierającą tylko te elementy dla których `function` zwróciło prawdę.

```
filter(lambda x: x and x % 2 == 0, [0,1,2,3,4,5,6,7])
```

Python3: :c

Przykład: pomiar temperatury

```
from collections import namedtuple
_T = namedtuple('_T', ['city', 'date', 'temp', 'scale'])

thermometers = [_T('Dortmund', '2013-03-1', 28, 'F'),
        _T('Paris', '2013-02-27', 5, 'C'), ... ]

min, max = None, None
for t in thermometers:
    if t.date != '2013-03-04':
        continue
    temp = t.temp if t.scale == 'C' else 5*(t.temp-32)/9
    if not min or temp < min:
        min = temp
    if not max or temp > max:
        max = temp
```

Przykład: pomiar temperatury

```
from collections import namedtuple
_T = namedtuple('_T', ['city', 'date', 'temp', 'scale'])

thermometers = [_T('Dortmund', '2013-03-1', 28, 'F'),
    _T('Paris', '2013-02-27', 5, 'C'), ... ]

today = filter(lambda t: t.date ==  '2013-03-04', thermometers)
celsius = map(lambda t: t.temp if t.scale == 'C' else 5*(t.temp-32)/9,
        today)
max = reduce(lambda a,b: a if a > b else b, celsius)
min = reduce(lambda a,b: a if a < b else b, celsius)
```

Przykład: pomiar temperatury

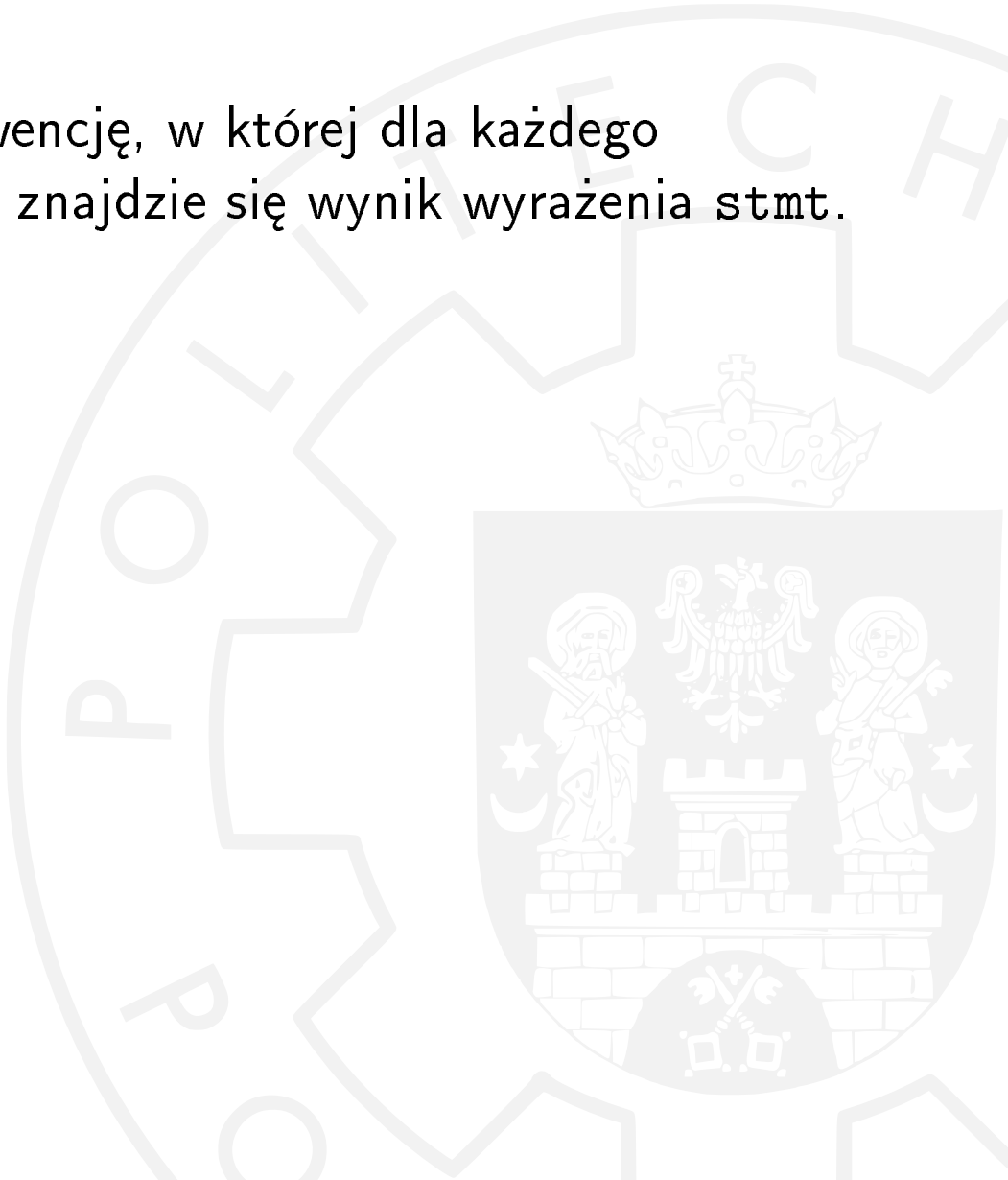
```
from collections import namedtuple
_T = namedtuple('_T', ['city', 'date', 'temp', 'scale'])

thermometers = [_T('Dortmund', '2013-03-1', 28, 'F'),
        _T('Paris', '2013-02-27', 5, 'C'), ... ]

filter(lambda t: t.date ==  '2013-03-04', thermometers)
map(lambda t: t.temp if t.scale == 'C' else 5*(t.temp-32)/9, _)
max = reduce(lambda a,b: a if a > b else b, _)
min = reduce(lambda a,b: a if a < b else b, _)
```

Generatory list

Mapowanie

```
[stmt for e in sequence]
```

Tworzy nową (anonimową) sekwencję, w której dla każdego elementu e z kolekcji `sequence` znajdzie się wynik wyrażenia `stmt`.



Generatory list

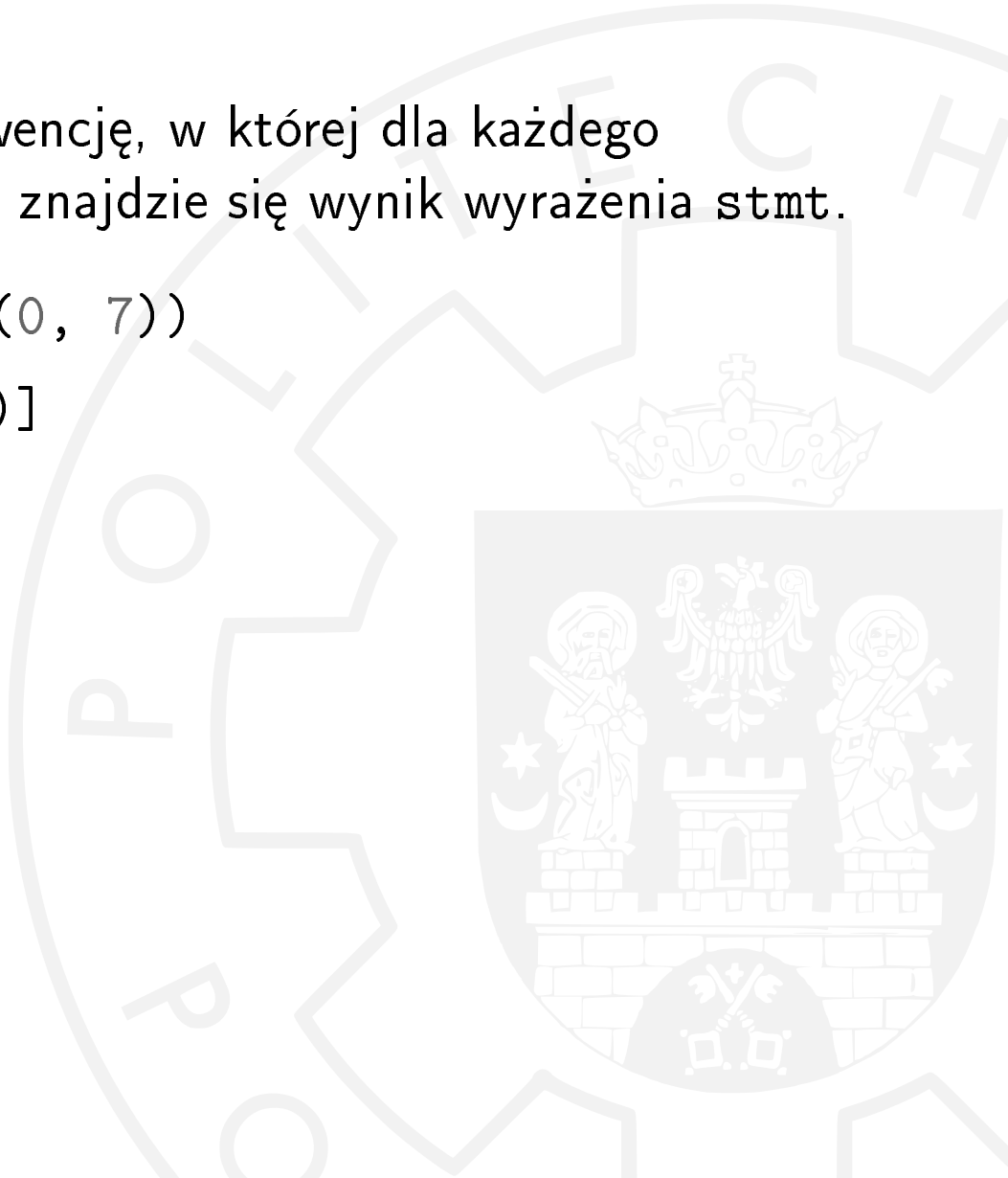
Mapowanie

```
[stmt for e in sequence]
```

Tworzy nową (anonimową) sekwencję, w której dla każdego elementu e z kolekcji sequence znajdzie się wynik wyrażenia stmt.

```
map(lambda x: 2**x, range(0, 7))
```

```
[2**x for x in range(0, 7)]
```



Generatory list

Mapowanie

```
[stmt for e in sequence]
```

Tworzy nową (anonimową) sekwencję, w której dla każdego elementu e z kolekcji sequence znajdzie się wynik wyrażenia stmt.

```
map(lambda x: 2**x, range(0, 7))
```

```
[2**x for x in range(0, 7)]
```

Filtrowanie

```
[stmt for e in sequence if cond]
```

Tworzy nową (anonimową) sekwencję, w której dla każdego elementu e z kolekcji sequence **dla którego jest spełniony warunek** cond znajdzie się wynik wyrażenia stmt.

Generatory list

Mapowanie

```
[stmt for e in sequence]
```

Tworzy nową (anonimową) sekwencję, w której dla każdego elementu e z kolekcji sequence znajdzie się wynik wyrażenia stmt.

```
map(lambda x: 2**x, range(0, 7))
```

```
[2**x for x in range(0, 7)]
```

Filtrowanie

```
[stmt for e in sequence if cond]
```

Tworzy nową (anonimową) sekwencję, w której dla każdego elementu e z kolekcji sequence **dla którego jest spełniony warunek** cond znajdzie się wynik wyrażenia stmt.

```
filter(lambda x: x and x % 2 == 0, range(0, 7))
```

```
[x for x in range(0, 7) if x and x % 2 == 0]
```

Przykład: pomiar temperatury

```
from collections import namedtuple
_T = namedtuple('_T', ['city', 'date', 'temp', 'scale'])

thermometers = [_T('Dortmund', '2013-03-1', 28, 'F'),
        _T('Paris', '2013-02-27', 5, 'C'), ... ]

filter(lambda t: t.date ==  '2013-03-04', thermometers)
map(lambda t: t.temp if t.scale == 'C' else 5*(t.temp-32)/9, _)
max = reduce(lambda a,b: a if a > b else b, _)
min = reduce(lambda a,b: a if a < b else b, _)
```

Przykład: pomiar temperatury

```
from collections import namedtuple
from functools import reduce
_T = namedtuple('_T', ['city', 'date', 'temp', 'scale'])

thermometers = [_T('Dortmund', '2013-03-1', 28, 'F'),
        _T('Paris', '2013-02-27', 5, 'C'), ... ]

[t for t in thermometers if t.date ==  '2013-03-04']
[t.temp if t.scale == 'C' else 5*(t.temp-32)/9 for t in _]
max = reduce(lambda a,b: a if a > b else b, _)
min = reduce(lambda a,b: a if a < b else b, _)
```

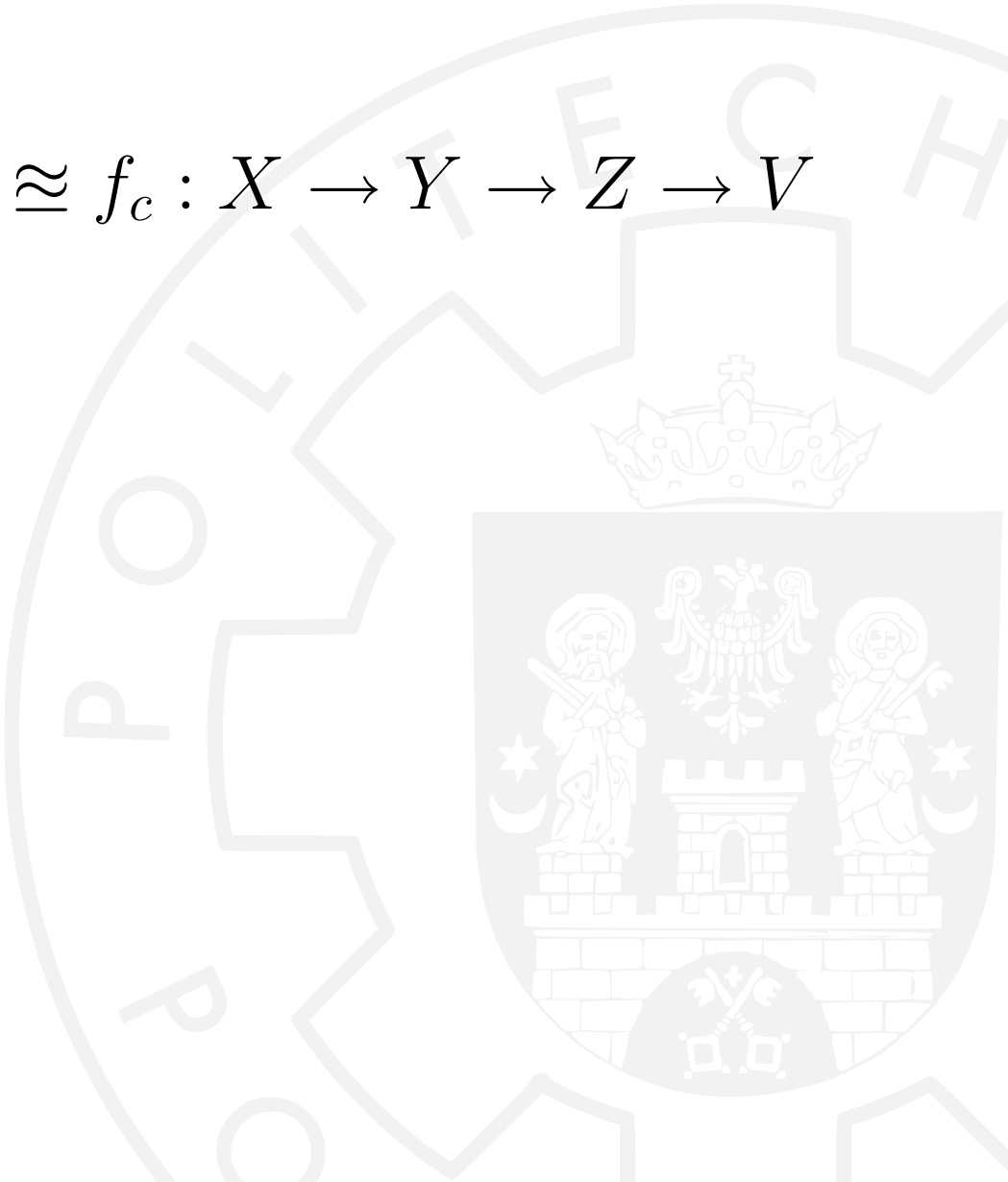
Currying

partial application

Currying

Schönfinkeling

$$f : (X \times Y \times Z) \to V \cong f_c : X \to Y \to Z \to V$$



Currying

Schönfinkeling

$$f : (X \times Y \times Z) \to V \cong f_c : X \to Y \to Z \to V$$

```
lambda x, y, z: v == lambda x: lambda y: lambda z: v
```

Currying

Schönfinkeling

$$f : (X \times Y \times Z) \to V \cong f_c : X \to Y \to Z \to V$$

```
lambda x, y, z: v == lambda x: lambda y: lambda z: v
```

częściowa aplikacja

`partial(function[, *args])`

Tworzy nową funkcję* z ustawionymi argumentami `args` na podstawie funkcji `function`.

Funkcja z pakietu `functools`.

Currying

Schönfinkeling

$$f : (X \times Y \times Z) \to V \cong f_c : X \to Y \to Z \to V$$

```
lambda x, y, z: v == lambda x: lambda y: lambda z: v
```

częściowa aplikacja

`partial(function[, *args])`

Tworzy nową funkcję* z ustawionymi argumentami args na podstawie funkcji `function`.

Funkcja z pakietu `functools`.

przykład

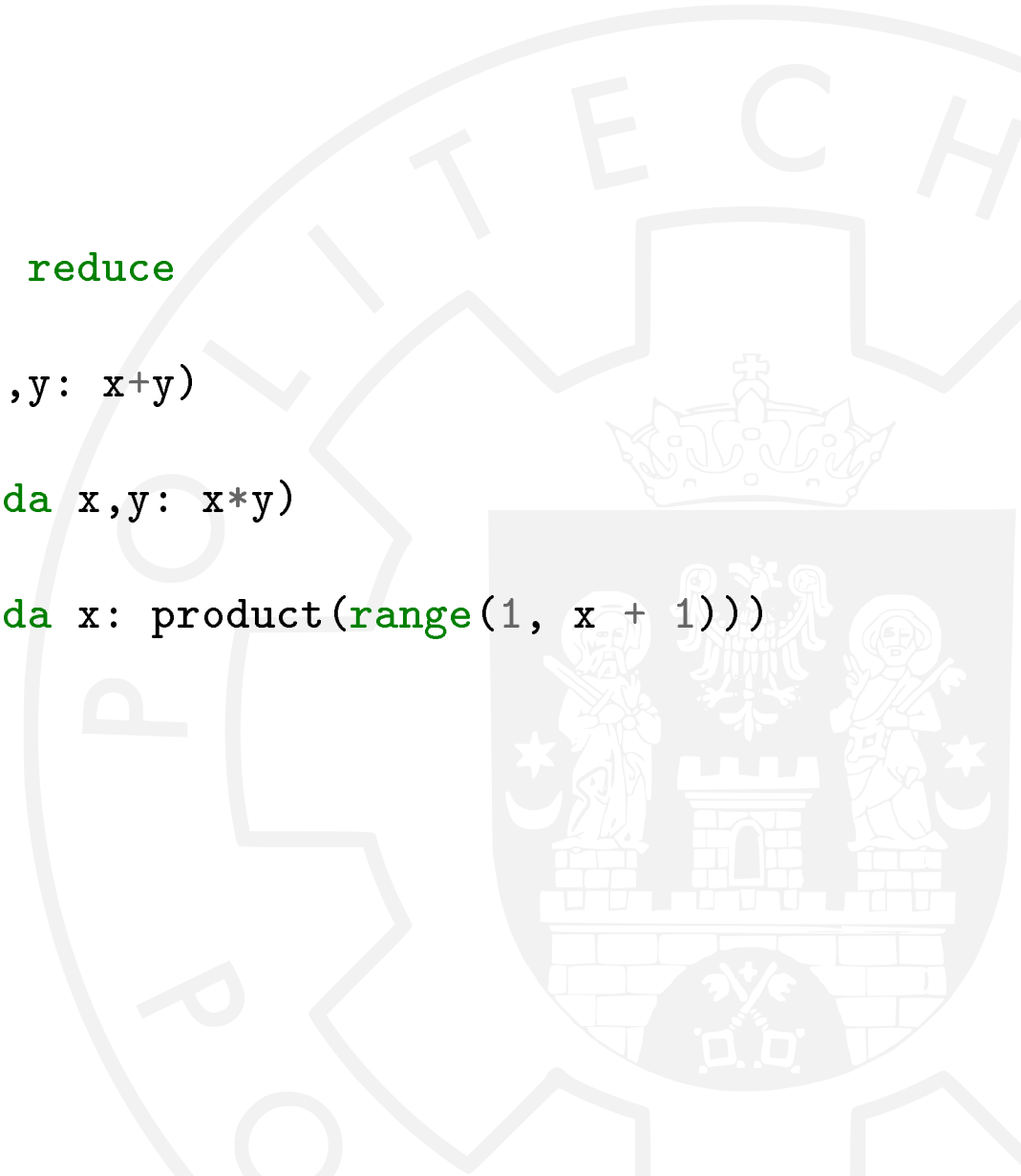
```
from functools import partial

power = lambda x, y: x**y
square = partial(power, y = 2)
cube = partial(power, y = 3)

print square(3), cube(3)
```

Przykład: funkcje na listach

```
from functools import partial, reduce

def sum(ls):
    return reduce(lambda x,y: x+y, ls)

def product(ls):
    return reduce(lambda x,y: x*y, ls)

def factorial(ls):
    return map(lambda x: product(range(1, x + 1)), ls)
```

Przykład: funkcje na listach

```
from functools import partial, reduce

sum = partial(reduce, lambda x,y: x+y)

product = partial(reduce, lambda x,y: x*y)

factorial =  partial(map, lambda x: product(range(1, x + 1)))
```



Przykład: numpad

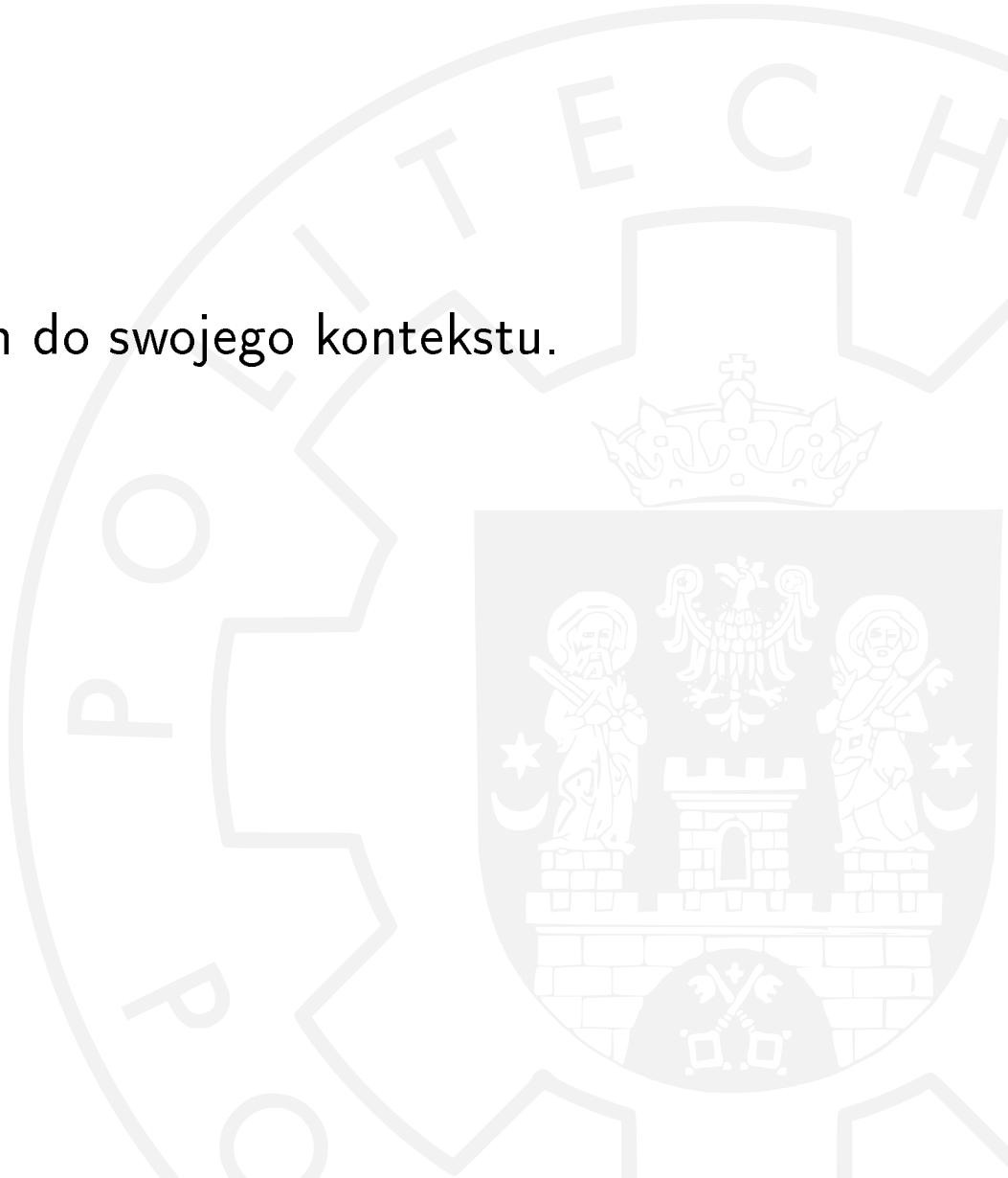
```
import Tkinter
from functools import partial

window = Tkinter.Tk()
contents = Tkinter.Variable(window)
display = Tkinter.Entry(window, textvariable=contents)
display.pack()

def clicked(digit):
    contents.set(contents.get() + str(digit))

for number in [7, 8, 9, 4, 5, 6, 1, 2, 3, 0]:
    button = Tkinter.Button(window, text=number,
                command=partial(clicked, number))

    button.pack(side="left", fill="x")

Tkinter.mainloop()
```

Domknięcie

closures

Domknięcie

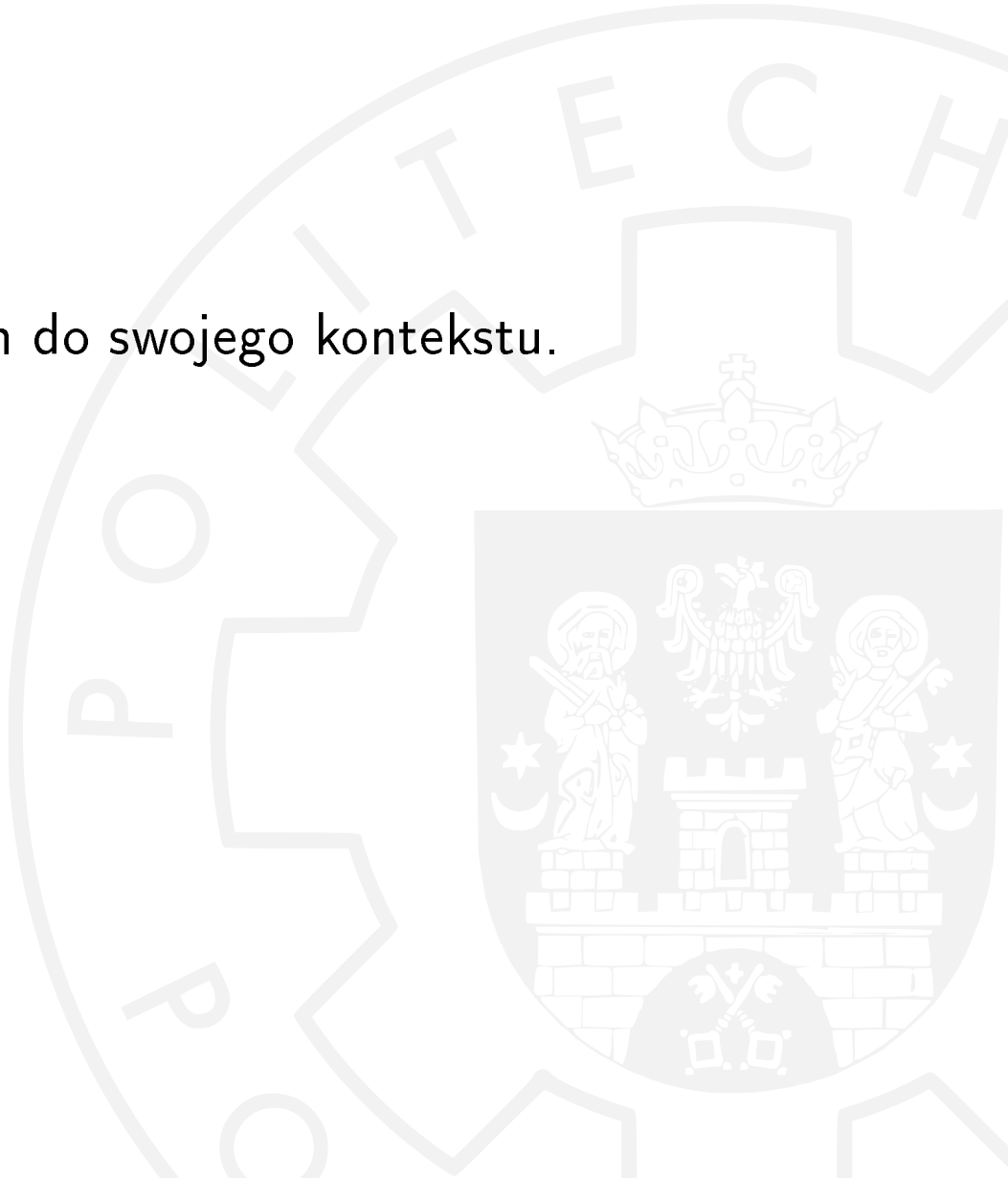
Funkcje z dostępem do swojego kontekstu.



Domknięcie

Funkcje z dostępem do swojego kontekstu.

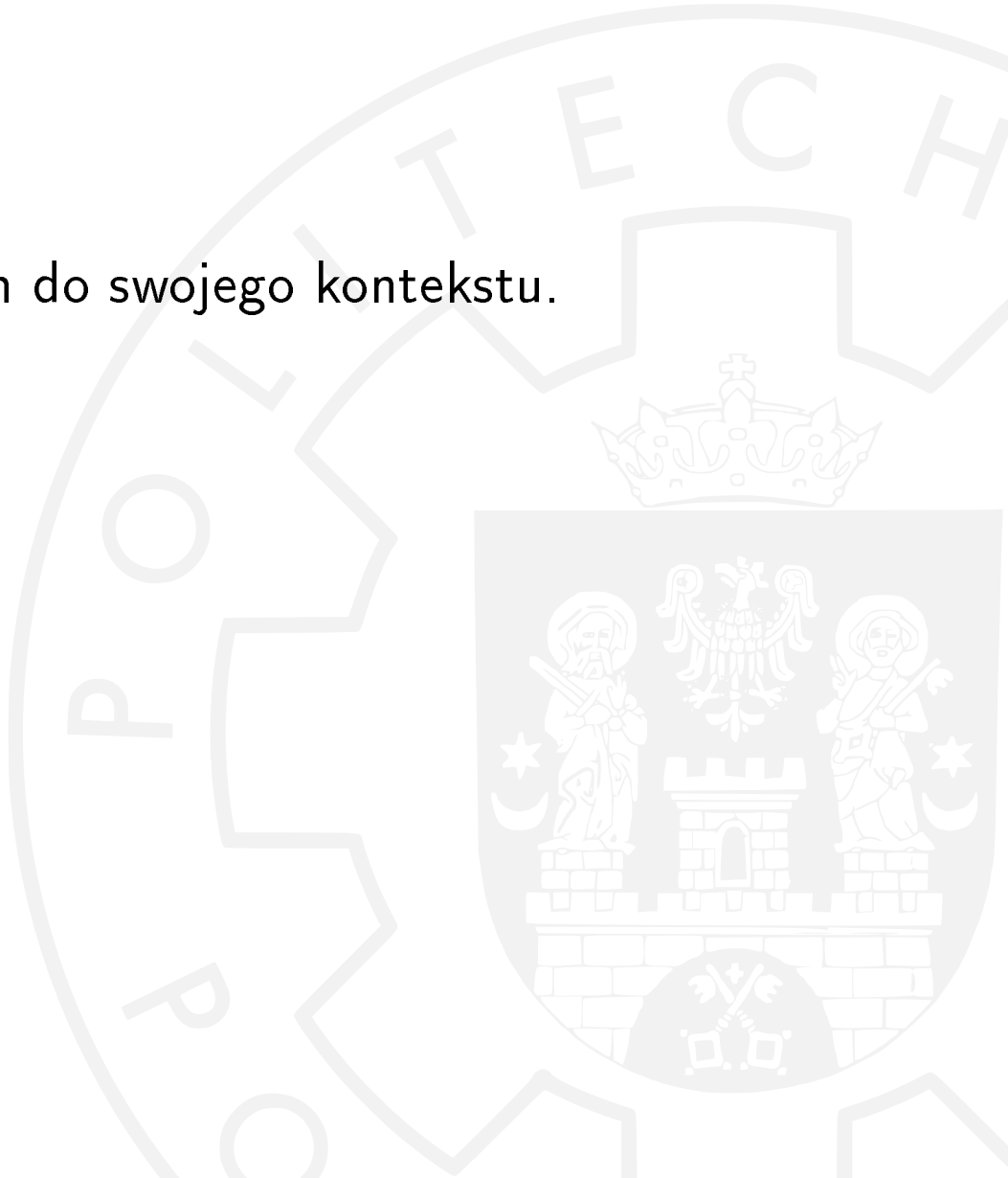
```
pi = 3.14
def area(r):
    return pi * r**2
```



Domknięcie

Funkcje z dostępem do swojego kontekstu.

```
def make_area():
    pi = 3.14
    def area(r):
        return pi * r**2
    return area
```

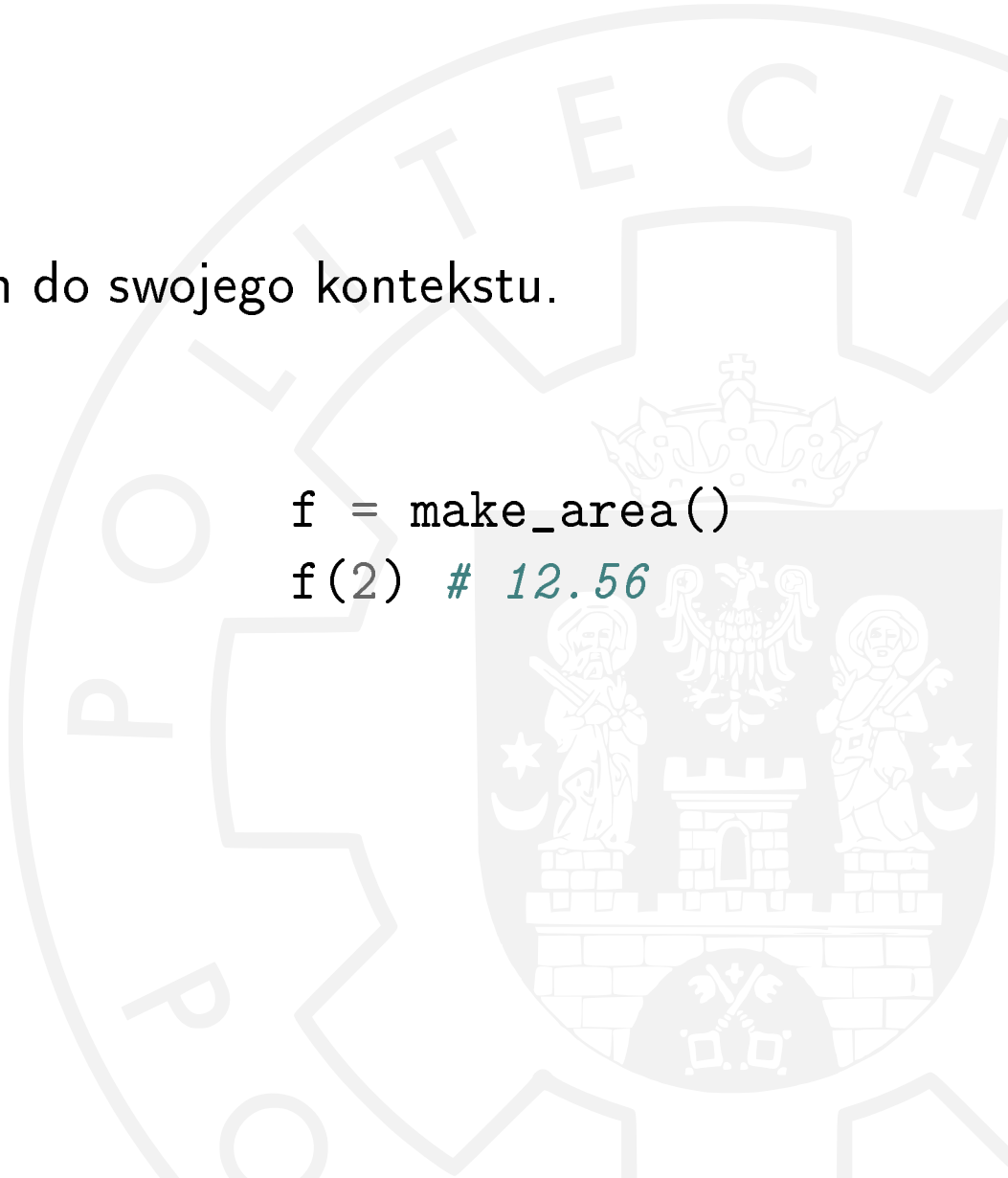


Domknięcie

Funkcje z dostępem do swojego kontekstu.

```
def make_area():
    pi = 3.14
    def area(r):
        return pi * r**2
    return area
```

```
f = make_area()
f(2) # 12.56
```



Domknięcie

Funkcje z dostępem do swojego kontekstu.

```
def make_area():
    pi = 3.14
    def area(r):
        return pi * r**2
    return area
```

```
f = make_area()
f(2) # 12.56
```

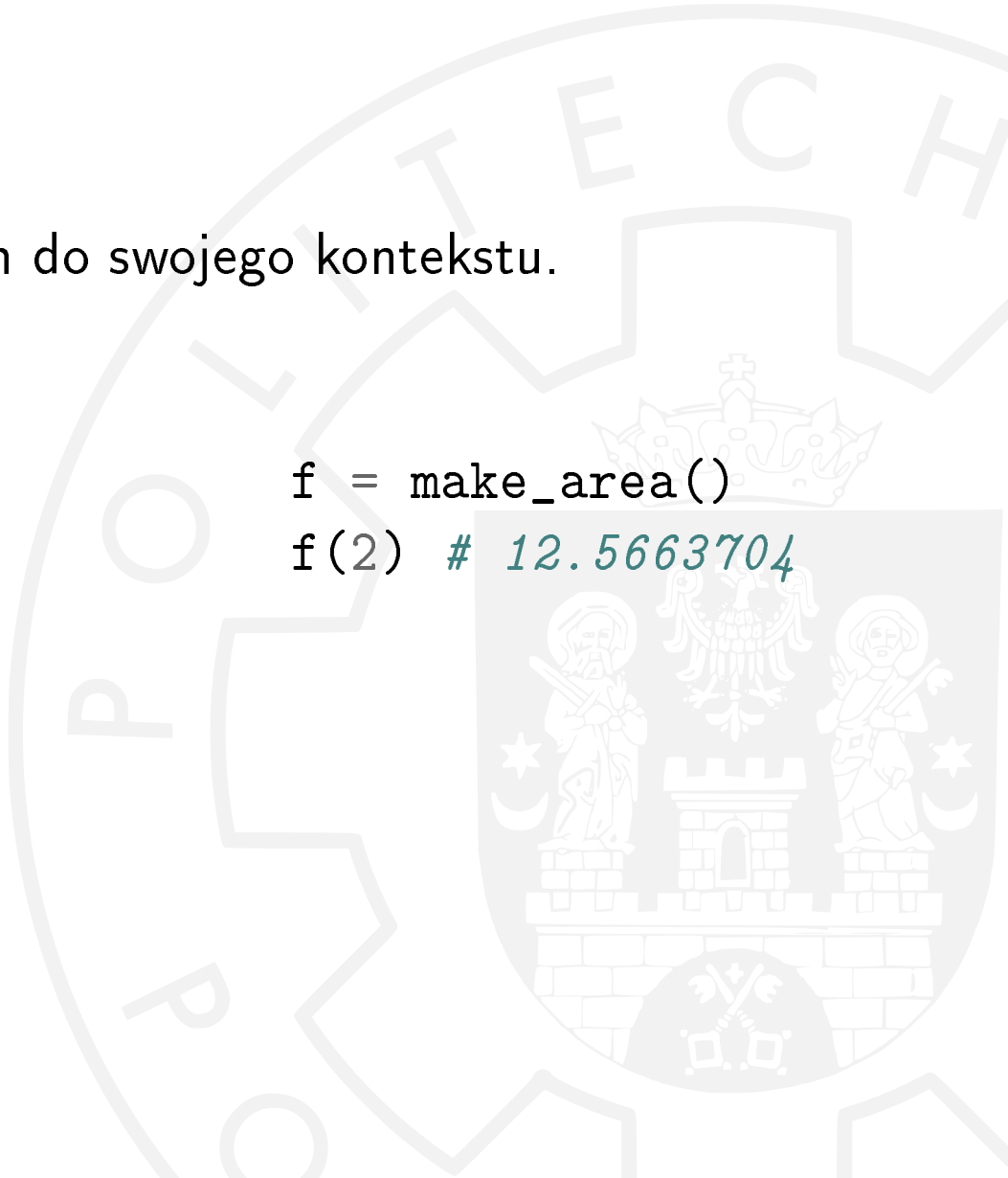
big deal...

Domknięcie

Funkcje z dostępem do swojego kontekstu.

```
def make_area():
    pi = 3.14
    def area(r):
        return pi * r**2
    pi = 3.1415926
    return area
```

```
f = make_area()
f(2) # 12.5663704
```



Przykład: liczniki

```
counter = 0
def next():
    global counter
    value = counter
    counter += 1
    return value
```

Przykład: liczniki

```
counter = 0
def next():
    global counter
    value = counter
    counter += 1
    return value
```

10 liczników?

Przykład: liczniki

```
def make_counter(start=0):
    counter = start
    def next():
        global counter
        value = counter
        counter += 1
        return value
    return next

counters = [make_counter() for _ in range(0, 10)]
```

Przykład: liczniki

```
def make_counter(start=0):
    counter = start
    def next():
        nonlocal counter
        value = counter
        counter += 1
        return value
    return next

counters = [make_counter() for _ in range(0, 10)]
```

Przykład: liczniki

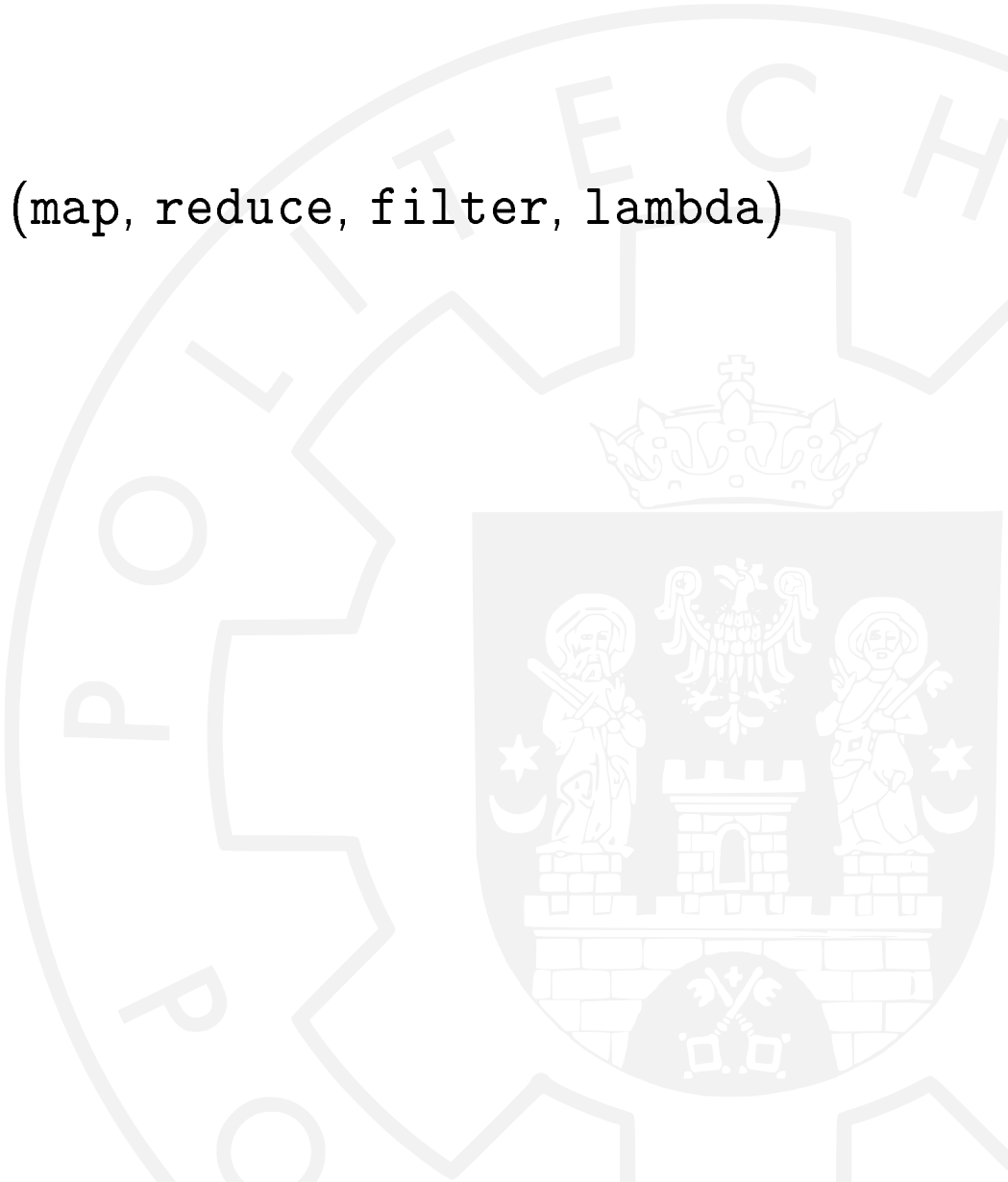
```
def make_counter(start=0):
    ctx = {'counter': start}
    def next():
        value = ctx['counter']
        ctx['counter'] += 1
        return value
    return next

counters = [make_counter() for _ in range(0, 10)]
```

Podsumowanie

the end is nigh

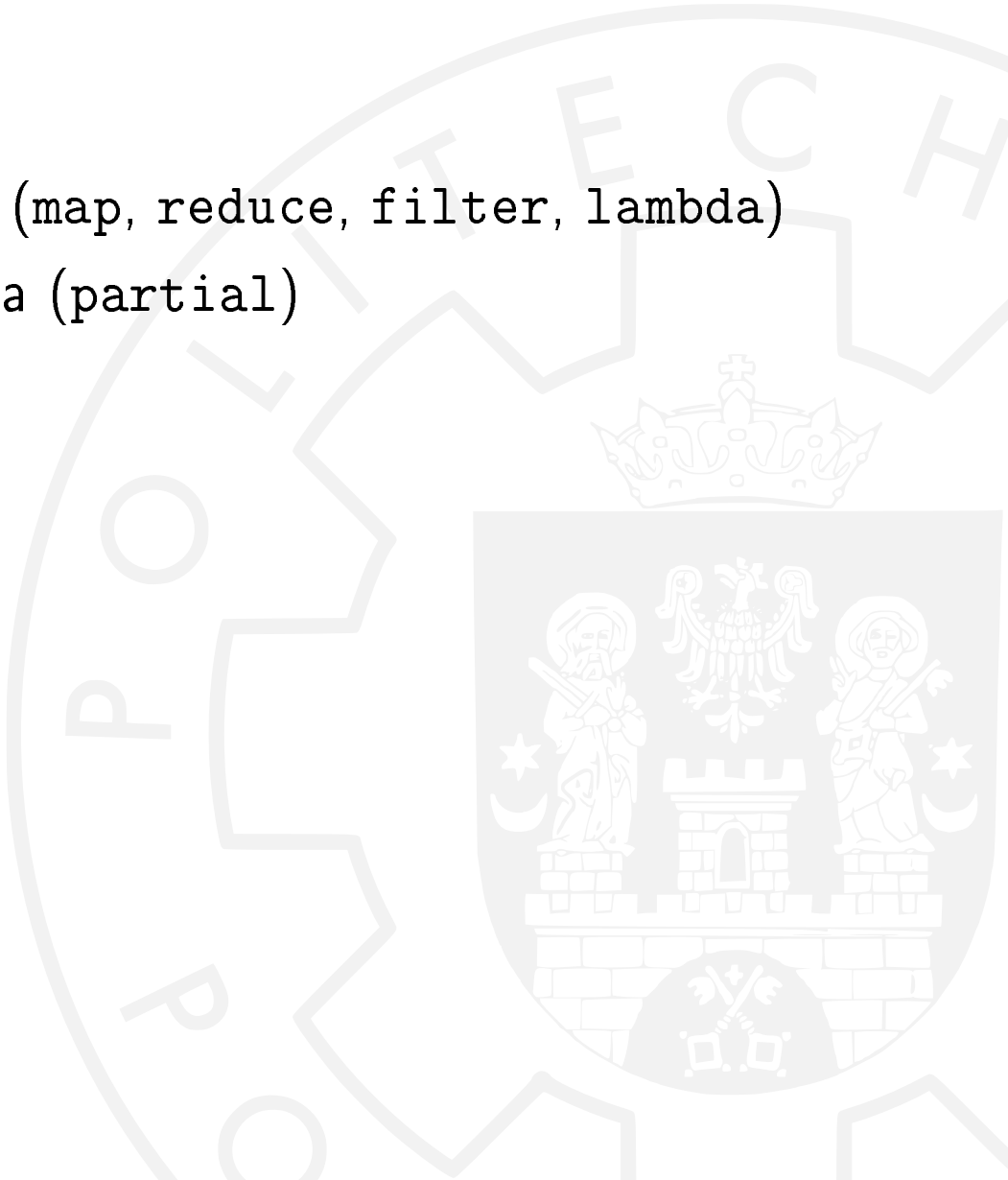
Zagadnienia

- funkcje pierwszej klasy itp. (`map`, `reduce`, `filter`, `lambda`)



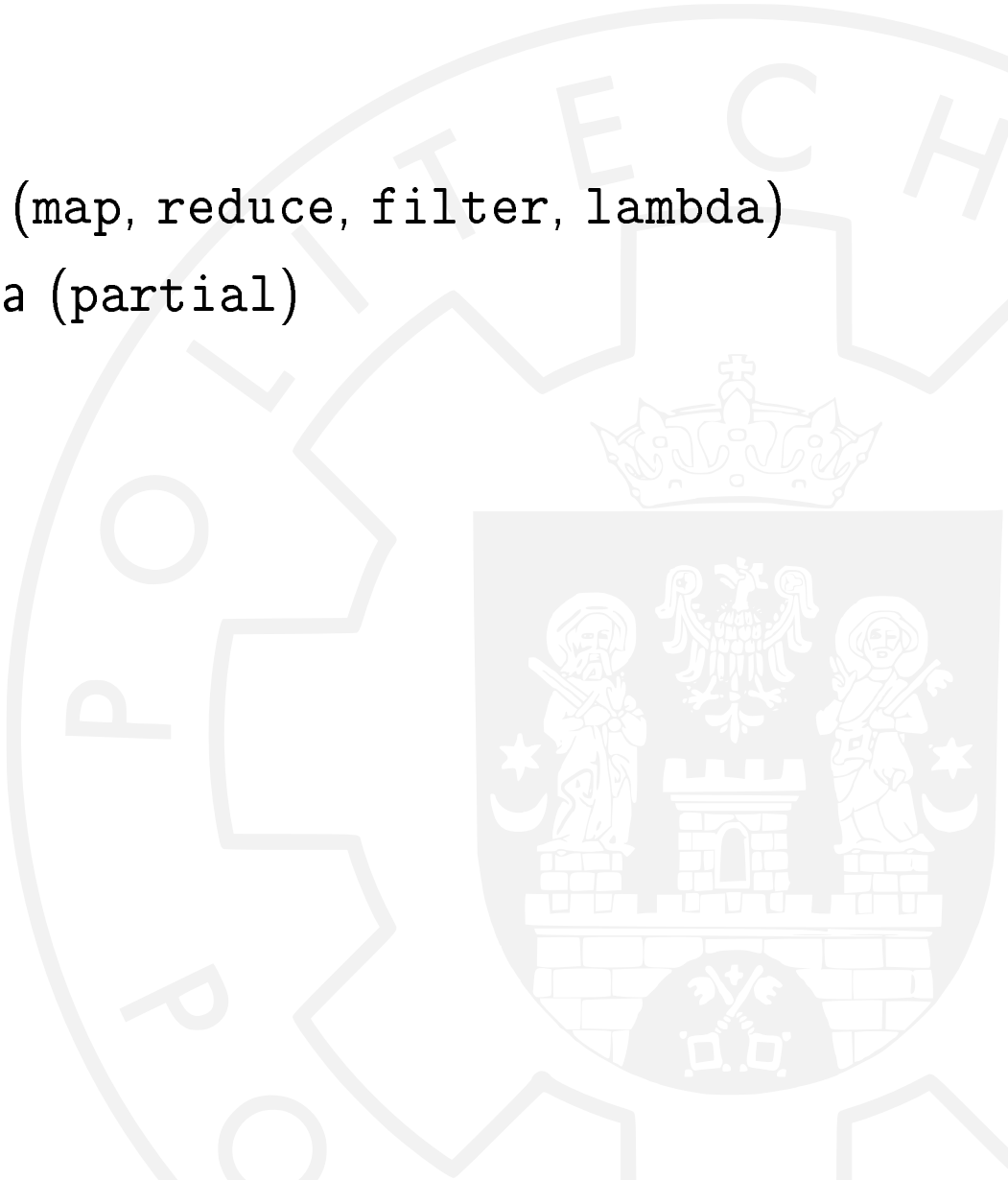
Zagadnienia

- funkcje pierwszej klasy itp. (`map`, `reduce`, `filter`, `lambda`)
- currying/częściowa aplikacja (`partial`)



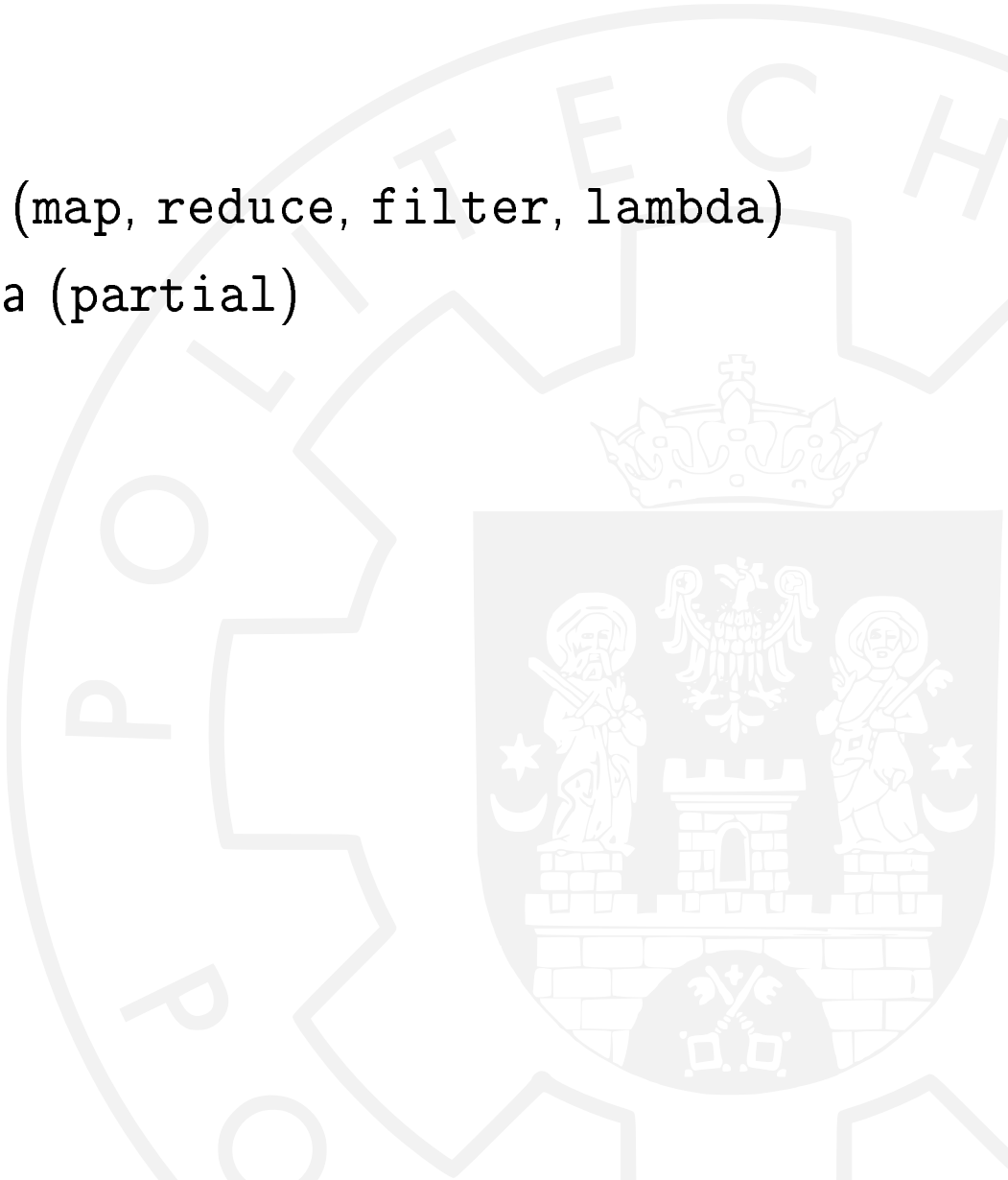
Zagadnienia

- funkcje pierwszej klasy itp. (`map`, `reduce`, `filter`, `lambda`)
- currying/częściowa aplikacja (`partial`)
- domknięcia



Zagadnienia

- funkcje pierwszej klasy itp. (`map`, `reduce`, `filter`, `lambda`)
- currying/częściowa aplikacja (`partial`)
- domknięcia

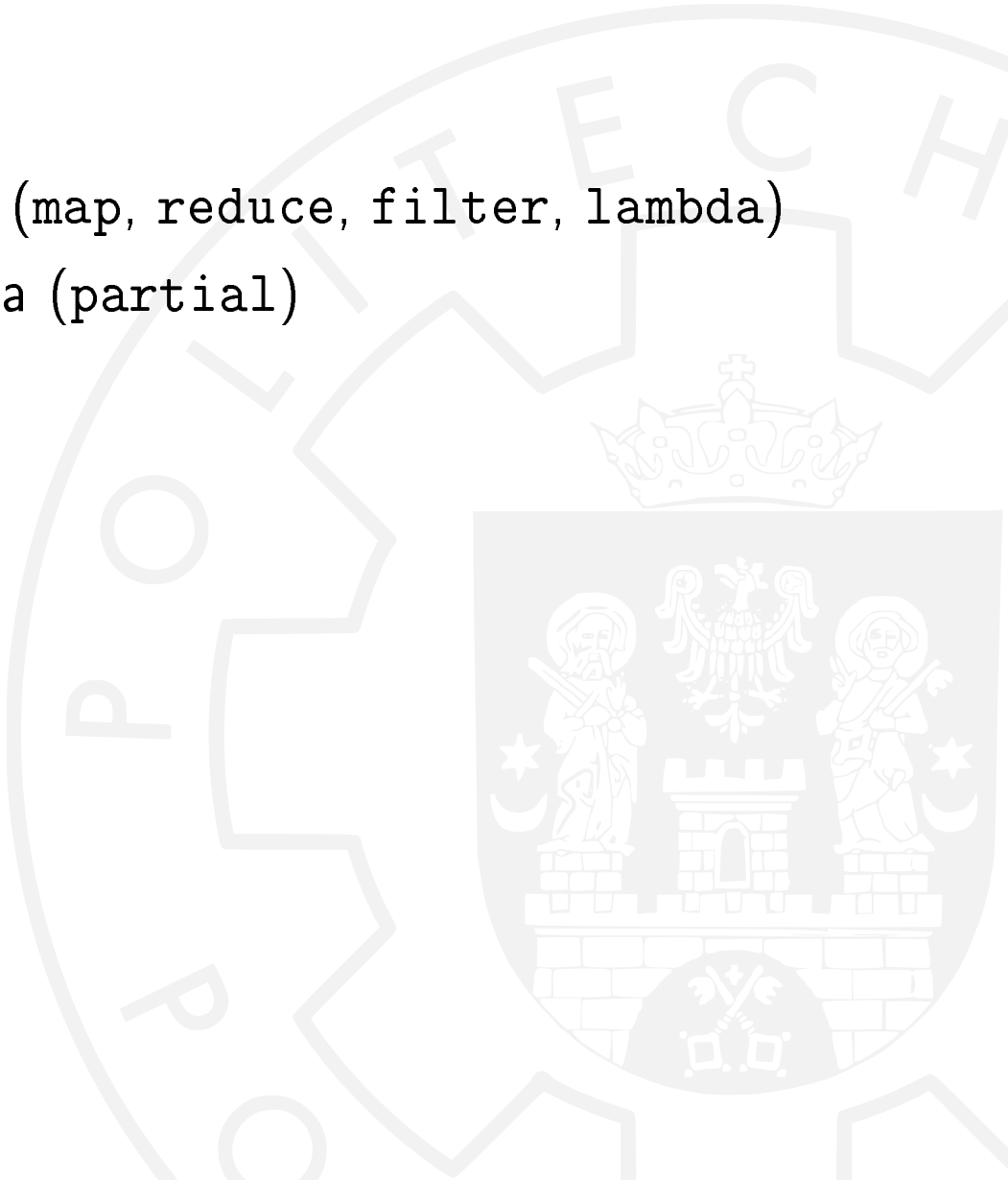


Zagadnienia

- funkcje pierwszej klasy itp. (`map`, `reduce`, `filter`, `lambda`)
- currying/częściowa aplikacja (`partial`)
- domknięcia

Why bother?

- prostsze rozwiązania

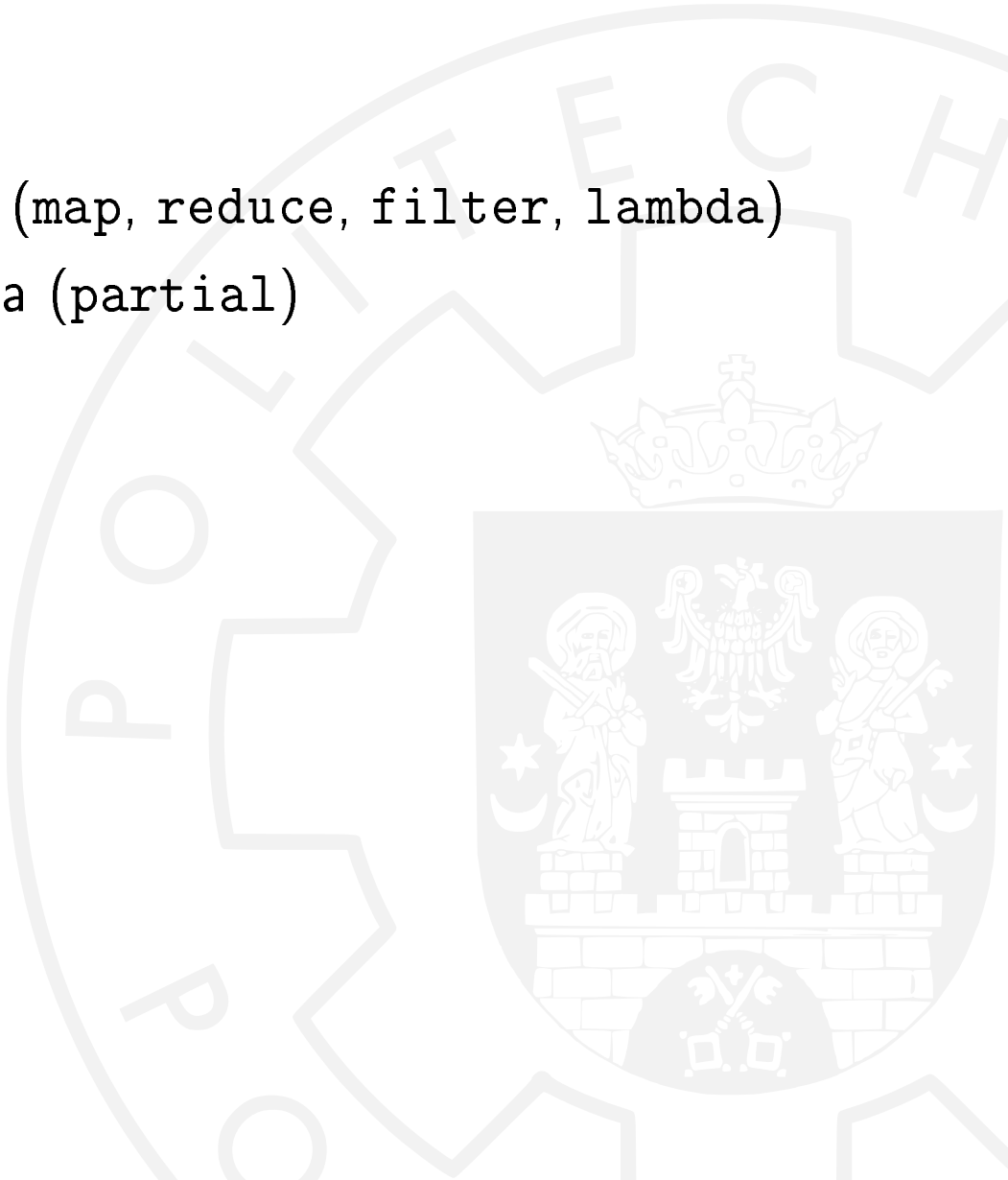


Zagadnienia

- funkcje pierwszej klasy itp. (`map`, `reduce`, `filter`, `lambda`)
- currying/częściowa aplikacja (`partial`)
- domknięcia

Why bother?

- prostsze rozwiązania
- zwięzłość

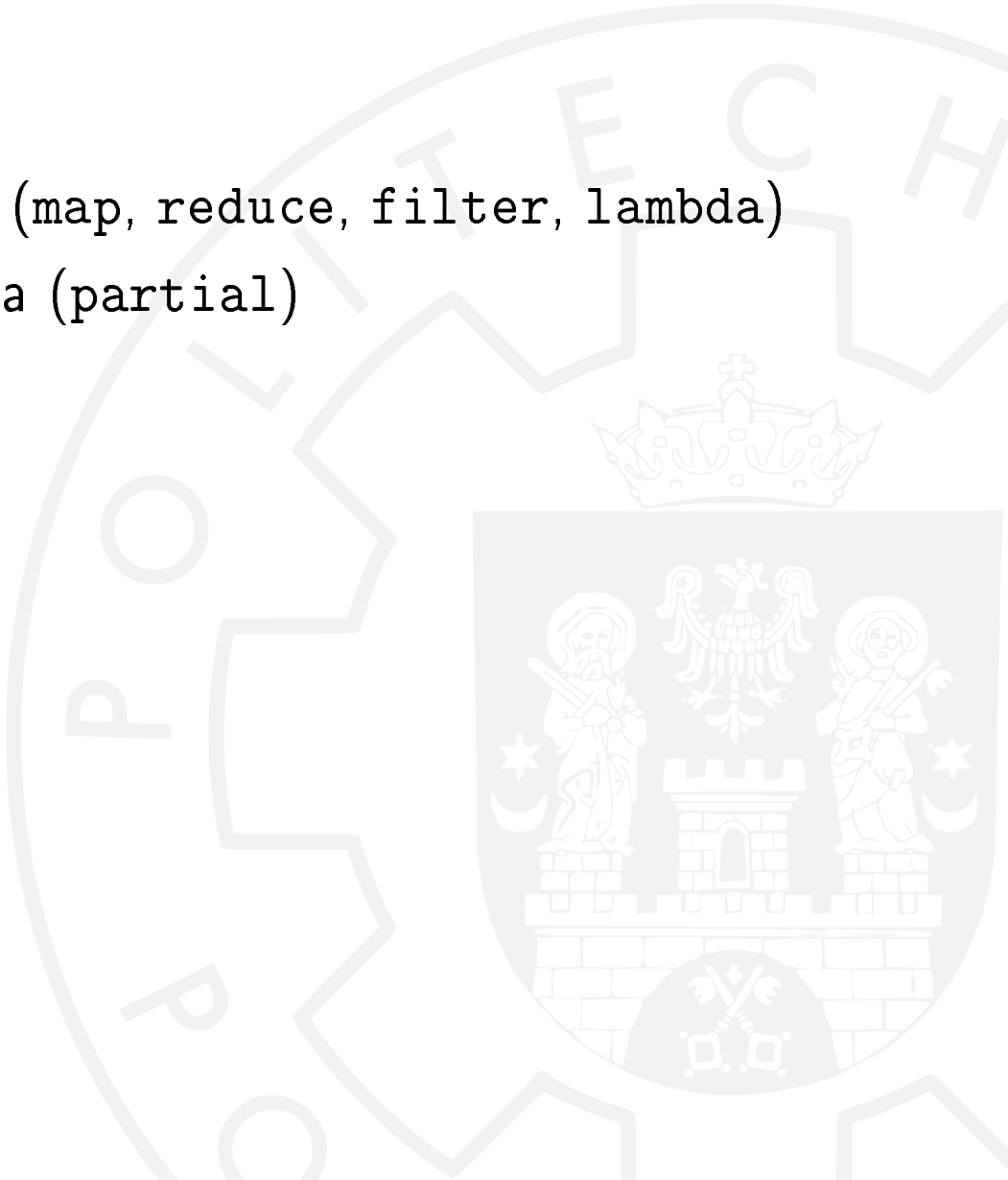


Zagadnienia

- funkcje pierwszej klasy itp. (`map`, `reduce`, `filter`, `lambda`)
- currying/częściowa aplikacja (`partial`)
- domknięcia

Why bother?

- prostsze rozwiązania
- zwięzłość
- geek cred



Pytania

Join us dsg.cs.put.poznan.pl/forum