

Python

- Funkcje 2 – generatory
- Klasy – co to jest i po co?
- Klasy w Pythonie
- Obiektowość

Generatory

- Zwykła funkcja zwraca wartości/obiekty podane za **return**
- W generatorach – zamiast **return** używa się **yield**
- Generatory opłacają się kiedy funkcja zwraca listę/krotkę (więcej niż jedną wartość)

Generator(2)

Po co?

- Oszczędzają pamięć i czas
- Program działa szybciej
- Nie oblicza się niepotrzebnych wartości

można skończyć na pierwszej, która spełnia określone warunki

Generator(3)

Składnia:

```
def nazwa_funkcji ( argumenty ):
```

```
→ ...
```

```
→ yield zwracana_wartość
```

Generator(4)

Przykład – funkcja podobna do range()
ale działa też na floatach

```
def frange( f, to, step ):  
    while f < to:  
        yield f  
        f += step
```

Klasy

- Klasa to podstawa obiektowości
- Jest "przepisem" na obiekty
- Obiekt = instancja klasy
- Klasa \sim typ

Klasy(2)

- DRY – don't repeat yourself
- KISS – keep it simple, stupid
- Oszczędzają pisanie i błędów

Klasy(3)

- Słowo kluczowe *class*
- Pola – wartości wewnątrz klasy
- Metody – funkcje klasy
- Metody ukryte: `__nazwametody__()`
podwójne podkreślenia

Klasy(4)

Przykład:

```
class klasa( list ): #dziedziczy po list
→ def __init__(self, *args):
→ → ...
→ → return ...
→ zmienna = ...
```

```
class klasa:
→ zmienna = cos
→ def metoda(self):
→ → self.zmienna = cos_innego
```

Klasy(5) – Dziedziczenie

```
class nowa_klasa1( ):
    pole = 'nic'
    def funk(self, argument):
        try:
            int(argument)
        except ValueError:
            return argument, 'to nie liczba'
        return argument**2
    def __init__(self, tekst):
        self.pole = tekst

ob1 = nowa_klasa1( 'cos')
```

Klasy(6) – Dziedziczenie

```
class klasa_potomna( nowa_klasa1 ):
    def ret_pole(self):
        return self.pole
```

```
dziecko = klasa_potomna( 'tekst' )
dziecko.ret_cos() # tekst
dziecko.funk(9) #zwraca 81
```