

# PODSTAWY PROGRAMOWANIA W PYTHON

Dzień 13





### AGENDA DAY 13

- type hinty
- dostarczanie argumentów do programu z konsoli
- •lambda funkcje bez nazwy
- generatory, komenda yield
- -dekoratory



## Type hints

Python "sugeruje" typy



## Type hints, czyli jak wspomóc IDE

```
def funkcja_a(arg1: str, arg2: dict, arg3: float = 4.0) -> list:
    arg1.format()
    arg2.items()
    arg3.is_integer()
    return []
```



## **Argumenty programu**

Jak przekazać coś podczas wywołania



## argumenty programów

```
python plik.py
python plik.py opcja1 opcja2 opcja3
import sys
sys.argy – lista argumentów przekazanych do Python
['plik.py', 'opcja1', 'opcja2', 'opcja3']
sys.argv[0] - ścieżka uruchomionego pliku
sys.argv[1:] - lista parametrów programu (jako stringi)
```



## Lambda

anonimowa funkcja



## Lambda definicja

```
wartość zwracana
x = lambda a:
kwadrat = x(4)
                  argumenty
policz = lambda zdanie, znak: zdanie.count(znak)
print(policz("ala ma kota", 'a'))
```



## Lambda przykład użycia

posortowani = sorted(pracownicy, key=lambda p: p["wiek"])



#### **Generator**

funkcja imitująca iterablę



## **Generatory** yield

```
def kwadraty_lista(lista_a):
    temp = []
    for x in lista_a:
        temp.append(x ** x)
    return temp
def kwadraty_generator(lista_a):
    for x in lista a:
```

Zwraca wartość zachowując stan wykonania funkcji, wracając do niego przy następnym wywołaniu.



#### **Generator comprehension**

```
kwadraty_lista = [x ** 2 \text{ for } x \text{ in } [1, 2, 3, 4, 5]]
kwadraty_generator = (x ** 2 \text{ for } x \text{ in } [1, 2, 3, 4, 5])
```



## **Context manager**

with - zautomatyzuj wejście i wyjście

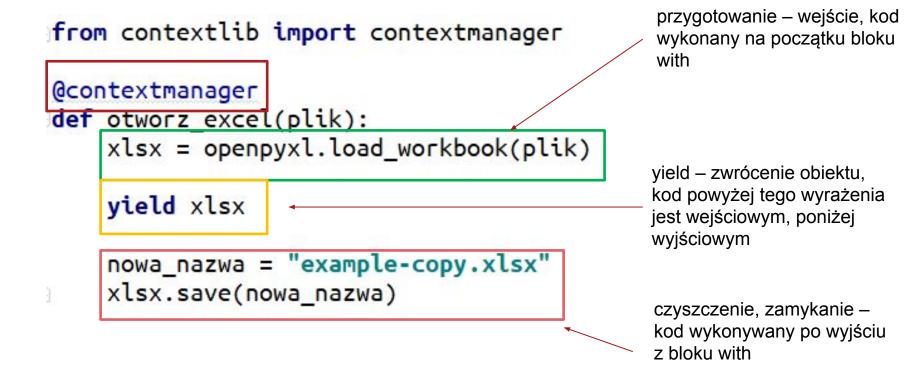


#### context managers przykład – bez menedżera

```
import openpyxl
plik = openpyxl.load_workbook("example.xlsx")
# coś robimy
#...
#
plik.save("example-copy.xlsx")
plik.close()
```



#### context managers przykład – definicja menadżera





## context managers użycie

```
from contextlib import contextmanager
                                                              przygotowanie – wejście, kod
import openpyxl
                                                              wykonany na początku bloku
                                                              with
@contextmanager
def otworz_excel(plik):
    xlsx = openpyxl.load_workbook(plik)
    yield xlsx
                                                              yield – zwrócenie obiektu,
    nowa_nazwa = "example-copy.xlsx"
                                                              kod powyżej tego wyrażenia
                                                              jest wejściowym, poniżej
    xlsx.save(nowa_nazwa)
                                                              wyjściowym
with otworz excel("example.xlsx") as plik excel:
                                                              czyszczenie, zamykanie –
    # cos robimy
                                                               kod wykonywany po wyjściu
    print(plik_excel.sheetnames)
                                                              z bloku with
```



## **Dekorator**

dodatkowa funkcjonalność do funkcji



#### dekoratory

```
funkcja "udekorowana",
                                                      to ją faktycznie będziemy
  nagłówek dekoratora, jako
                                                      wywoływać
  parametr przyjmuje funkcję
         def logger (func):
                                     **kwargs)
              def inner(*args,
                        nt(f"Arguments were: {args}, {kwargs}")
                    return func(*args, **kwargs)
              return inner
                                                        wywołanie funkcji
dekorator zwraca obiekt
                                                        "dekorowanej", ważne jest,
definicji "udekorowanej"
                                                        aby przekazać parametry
funkcji
```

http://simeonfranklin.com/blog/2012/jul/1/python-decorators-in-12-steps/





## Thanks!!