Wprowadzenie do języka Python - Wykład 7

Obsługa plików

https://docs.python.org/3/library/io.html

```
1 with open('plik.txt', 'r') as f:
2     print(f.read())
```

```
1 f = open('plik.txt', 'r')
2 print(f.read())
3 f.close()
```

```python
# Odczyt
with open('plik.txt', 'r') as f:
    for line in f:
        print(line.strip())

# Zapis
with open('plik.txt', 'w') as f:
    f.write("Hello, world!\n")
    f.write("Welcome to Python IO.\n")
```

```python
# Odczyt pliku binarnego
with open('obraz.jpg', 'rb') as f:
    data = f.read()

# Zapis pliku binarnego
with open('kopia_obrazu.jpg', 'wb') as f:
    f.write(data)
```

```python
try:
    with open('nieistniejacy_plik.txt', 'r') as f:
        print(f.read())
except FileNotFoundError:
    print("Plik nie istnieje.")
except PermissionError:
    print("Brak uprawnień do odczytu pliku.")
except Exception as e:
    print(f"Wystąpił nieoczekiwany błąd: {e}")
```

Tryby dostępu

- ‘b’ - binarny (binary) Dodawany do jednego z powyższych trybów, wskazuje, że plik ma być obsługiwany jako plik binarny. W przypadku plików tekstowych, tryb binarny nie jest dodawany.
- ‘t’ - tekstowy (text) Oznacza, że plik ma być obsługiwany jako plik tekstowy. Jest to domyślne zachowanie, jeśli nie podano trybu binarnego.
- ‘+’ - aktualizacja (update) Dodawany do jednego z powyższych trybów (oprócz ‘x’), umożliwia jednoczesny odczyt i zapis pliku.

Kombinacje trybów:

- ‘rt’ lub ‘r’: Odczyt pliku tekstowego.
- ‘wt’ lub ‘w’: Zapis pliku tekstowego (nadpisuje istniejący plik lub tworzy nowy).
- ‘at’ lub ‘a’: Dopisywanie do pliku tekstowego.
- ‘xt’ lub ‘x’: Tworzenie nowego pliku tekstowego (zwraca błąd, jeśli plik istnieje).

- ‘rb’: Odczyt pliku binarnego.
- ‘wb’: Zapis pliku binarnego (nadpisuje istniejący plik lub tworzy nowy).
- ‘ab’: Dopisywanie do pliku binarnego.
- ‘xb’: Tworzenie nowego pliku binarnego (zwraca błąd, jeśli plik istnieje).

- ‘r+t’ lub ‘r+’: Odczyt i zapis pliku tekstowego.
- ‘w+t’ lub ‘w+’: Zapis i odczyt pliku tekstowego (nadpisuje istniejący plik lub tworzy nowy).
- ‘a+t’ lub ‘a+’: Dopisywanie i odczyt pliku tekstowego.
- ‘x+t’ lub ‘x+’: Tworzenie i odczyt/zapis nowego pliku tekstowego

Jak wybrać?

- Czytanie czy zapisywanie?
- Plik tekstowy czy binarny?
- Czy potrzebujesz jednocześnie odczytywać i zapisywać plik?
- Błędy i obsługa wyjątków

1. IOBase: `IOBase` to klasa bazowa dla wszystkich klas strumieni I/O w module `io`. Definiuje podstawowe metody, takie jak `close()`, `flush()`, `seek()`, `tell()` oraz `__enter__()` i `__exit__()`, które umożliwiają korzystanie z bloków `with`. Ta klasa nie powinna być używana bezpośrednio, ale stanowi podstawę dla bardziej wyspecjalizowanych klas.

2. RawIOBase: `RawIOBase` to klasa bazowa dla surowych (raw) strumieni I/O, które obsługują operacje na niskim poziomie. Strumienie surowe operują na bajtach i nie korzystają z buforowania. Ta klasa rozszerza `IOBase` i implementuje dodatkowe metody, takie jak `read()`, `readall()`, `readinto()`, `write()` oraz metody operujące na plikach, takie jak `truncate()` czy `seek()`. Klasy pochodne od `RawIOBase` są używane, gdy konieczne jest bezpośrednie czytanie lub zapisywanie danych binarnych z/do urządzenia I/O (np. plików, gniazd sieciowych).

3. BufferedIOBase: `BufferedIOBase` to klasa bazowa dla buforowanych strumieni I/O. Buforowanie polega na tym, że dane są przechowywane w pamięci podręcznej, zanim zostaną przesłane do urządzenia I/O lub z niego odczytane. Buforowanie pomaga poprawić wydajność strumieni I/O, szczególnie gdy operacje odczytu/zapisu są małe lub wykonywane w sposób nieciągły. Klasa `BufferedIOBase` rozszerza klasę `IOBase` i definiuje dodatkowe metody, takie jak `read()`, `readinto()`, `write()` oraz `detach()`. Klasy pochodne od `BufferedIOBase` obejmują `BufferedReader`, `BufferedWriter` oraz `BufferedRandom`, które implementują buforowane strumienie binarne.

TextIOBase: `TextIOBase` to klasa bazowa dla strumieni tekstowych I/O. Strumienie tekstowe działają na poziomie znaków, a nie bajtów, i zajmują się dekodowaniem i kodowaniem danych w trakcie odczytu i zapisu. Klasa `TextIOBase` rozszerza klasę `IOBase` i definiuje dodatkowe metody, takie jak `read()`, `readline()`, `readlines()`, `write()`, `writelines()` oraz `detach()`. Klasa `TextIOWrapper` dziedziczy po klasie `TextIOBase` i implementuje strumień tekstowy z użyciem buforowanego strumienia binarnego jako źródła danych. Strumienie tekstowe są wykorzystywane w przypadku pracy z plikami tekstowymi, gdzie zachodzi konieczność obsługi kodowania znaków.

```
with open('example.txt', 'r') as file:
    file_descriptor = file.fileno()
    print(f'Deskryptor pliku dla example.txt: {file_descriptor}')
```

```
with open('plik.txt', 'w') as file:
    file.write('Witaj, świecie!')
    file.flush()
    print('Zawartość bufora została zapisana na dysku.')
```

```
1  import sys
2
3  if sys.stdin.isatty():
4      print('Standardowy strumień wejścia jest powiązany z interaktywnym term
5  else:
6      print('Standardowy strumień wejścia nie jest powiązany z interaktywnym
7
8  if sys.stdout.isatty():
9      print('Standardowy strumień wyjścia jest powiązany z interaktywnym term
10 else:
11     print('Standardowy strumień wyjścia nie jest powiązany z interaktywnym
```

```
with open('plik.txt', 'r') as file:
    if file.readable():
        content = file.read()
        print('Zawartość pliku example.txt:')
        print(content)
    else:
        print('Plik example.txt nie jest do odczytu.')
```

```
1 with open('tekst.txt', 'r') as plik:
2     pierwsza_linia = plik.readline()
3     print(pierwsza_linia)
```

```
with open('plik.txt') as f:
    lines = f.readlines()
    for line in lines:
        print(line)

```

```python
with open('plik.txt', 'rb') as file:
    # Przesuń wskaźnik pliku na początek pliku (domyślnie)
    file.seek(0)

    # Odczytaj i wyświetl pierwszych 5 bajtów
    print(file.read(5))

    # Przesuń wskaźnik pliku 10 bajtów od początku pliku
    file.seek(10)

    # Odczytaj i wyświetl kolejnych 5 bajtów
    print(file.read(5))

    # Przesuń wskaźnik pliku 5 bajtów od bieżącej pozycji (whence=1)
    file.seek(5, 1)

    # Odczytaj i wyświetl kolejnych 5 bajtów
    print(file.read(5))
```

```python
with open('plik.txt', 'r') as file:
    # Sprawdź pozycję wskaźnika pliku na początku pliku
    position = file.tell()
    print(f'Aktualna pozycja wskaźnika pliku: {position}')

    # Odczytaj pierwszych 5 bajtów
    file.read(5)

    # Sprawdź pozycję wskaźnika pliku po odczytaniu 5 bajtów
    position = file.tell()
    print(f'Aktualna pozycja wskaźnika pliku: {position}')

    # Przesuń wskaźnik pliku 10 bajtów od początku pliku
    file.seek(10)

    # Sprawdź pozycję wskaźnika pliku po przesunięciu
    position = file.tell()
    print(f'Aktualna pozycja wskaźnika pliku: {position}')
```

```python
with open('plik.txt', 'r+') as file:
    # Odczytaj i wyświetl zawartość pliku przed skróceniem
    file.seek(0)
    content = file.read()
    print(f'Zawartość pliku przed skróceniem: {content}')

    # Przesuń wskaźnik pliku 10 bajtów od początku pliku
    file.seek(10)

    # Skróć plik do bieżącej pozycji wskaźnika pliku (10 bajtów)
    file.truncate()

    # Odczytaj i wyświetl zawartość pliku po skróceniu
    file.seek(0)
    content = file.read()
    print(f'Zawartość pliku po skróceniu: {content}')
```

```
lines = ['aaa\n', 'bbb\n', 'ccc\n']

with open('example.txt', 'w') as file:
    file.writelines(lines)

with open('example.txt', 'r') as file:
    content = file.read()
    print('Zawartość pliku example.txt:')
    print(content)
```

```python
import io

class ExampleRawIO(io.RawIOBase):
    def __init__(self, data):
        self.data = data.encode('utf-8')
        self.index = 0

    def readinto(self, b):
        read_length = min(len(b), len(self.data) - self.index)
        b[:read_length] = self.data[self.index:self.index+read_length]
        self.index += read_length
        return read_length

data = "To jest przykładowy tekst."

# Tworzymy obiekt typu ExampleRawIO (RawIOBase) z przykładowymi danymi
example_raw_io = ExampleRawIO(data)

# Odczytujemy wszystkie dane za pomocą metody readall()
```

```python
import io

class ExampleRawIO(io.RawIOBase):
    def __init__(self, data):
        self.data = data.encode('utf-8')
        self.index = 0

    def readinto(self, b):
        read_length = min(len(b), len(self.data) - self.index)
        b[:read_length] = self.data[self.index:self.index+read_length]
        self.index += read_length
        return read_length

data = "To jest przykładowy tekst."

# Tworzymy obiekt typu ExampleRawIO (RawIOBase) z przykładowymi danymi
example_raw_io = ExampleRawIO(data)

# Tworzymy bufor do przechowywania odczytanych danych
```

Moduł `itertools`

https://docs.python.org/3/library/itertools.html

Moduł itertools to część standardowej biblioteki Pythona, która oferuje zestaw szybkich, wydajnych narzędzi do pracy z iteratorami. Iterator to obiekt, który umożliwia przeglądanie kolejnych elementów kolekcji (np. listy, krotki, słownika) w sposób sekwencyjny. Moduł itertools jest szczególnie użyteczny w przypadku operacji na dużych zbiorach danych lub sekwencjach nieskończonych, gdyż pozwala na oszczędzenie pamięci poprzez pracę z danymi w sposób leniwy (ang. lazy).

```python
import itertools

# nieskończony iterator zaczynający się od 1
counter = itertools.count(1)

# pobieranie kolejnych wartości z iteratora
print(next(counter))  # 1
print(next(counter))  # 2
print(next(counter))  # 3
```

```python
import itertools

# Utworzenie iteratora, który zaczyna się od liczby 5 i ma krok 3
counter = itertools.count(start=5, step=3)

# Wyświetlenie pierwszych 5 liczb z iteratora
for i in range(5):
    print(next(counter))
```

```python
import itertools

# iterator cykliczny powtarzający elementy z listy
colors = itertools.cycle(['red', 'green', 'blue'])

# pobieranie kolejnych elementów z iteratora
print(next(colors))  # 'red'
print(next(colors))  # 'green'
print(next(colors))  # 'blue'
print(next(colors))  # 'red'
print(next(colors))  # 'green'
print(next(colors))  # 'blue'
```

```python
import itertools

# Sekwencja, którą chcemy powtarzać cyklicznie
sequence = ['A', 'B', 'C']

# Utworzenie iteratora, który cyklicznie powtarza elementy sekwencji
cycler = itertools.cycle(sequence)

# Wyświetlenie pierwszych 10 elementów z iteratora
for i in range(10):
    print(next(cycler))
```

```python
import itertools

# nieskończony iterator zawsze zwracający wartość 42
repeater = itertools.repeat(42)

# pobieranie kolejnych wartości z iteratora
print(next(repeater))  # 42
print(next(repeater))  # 42
print(next(repeater))  # 42
# itd.
```

```python
import itertools

# Obiekt, który chcemy powtarzać
object_to_repeat = "Olsztyn"

# Utworzenie iteratora, który powtarza obiekt 5 razy
repeater = itertools.repeat(object_to_repeat, times=5)

# Wyświetlenie elementów z iteratora
for item in repeater:
    print(item)
```

```python
import itertools

# iterator skumulowanych sum na liście [1, 2, 3, 4, 5]
cumulative_sums = itertools.accumulate([1, 2, 3, 4, 5])

# pobieranie kolejnych wartości z iteratora zwracającego skumulowane sumy
print(list(cumulative_sums))  # [1, 3, 6, 10, 15]

# iterator skumulowanych iloczynów na liście [1, 2, 3, 4, 5]
cumulative_products = itertools.accumulate([1, 2, 3, 4, 5], lambda x, y: x

# pobieranie kolejnych wartości z iteratora zwracającego skumulowane iloczy
print(list(cumulative_products))  # [1, 2, 6, 24, 120]
```

```python
import itertools

# łączenie trzech list w jedną
letters = ['a', 'b', 'c']
digits = [1, 2, 3]
symbols = ['!', '@', '#']
combined = itertools.chain(letters, digits, symbols)

# pobieranie kolejnych elementów z łączonego iteratora
for item in combined:
    print(item)
```

```python
import itertools

# lista zawierająca kilka list
nested = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]

# łączenie list zagnieżdżonych w jedną listę z użyciem chain.from_iterable(
flat = itertools.chain.from_iterable(nested)

# pobieranie kolejnych elementów z łączonego iteratora
for item in flat:
    print(item)
```

```python
import itertools

# lista zawierająca losowe wartości
data = [1, 2, 3, 4, 5, 6]

# lista zawierająca wartości boolowskie
selector = [True, False, True, False, True, False]

# filtracja elementów z użyciem compress()
filtered = itertools.compress(data, selector)

# wyświetlanie przefiltrowanych elementów
for item in filtered:
    print(item)
```

```python
import itertools

# lista zawierająca liczby parzyste i nieparzyste
numbers = [2, 4, 6, 9, 8, 10, 11, 12]

# funkcja lambda sprawdzająca, czy liczba jest parzysta
is_even = lambda x: x % 2 == 0

# pomijanie elementów początkowej sekwencji z użyciem dropwhile()
filtered = itertools.dropwhile(is_even, numbers)

# wyświetlanie pominiętych elementów
for item in filtered:
    print(item)
```

```python
import itertools

# lista zawierająca liczby parzyste i nieparzyste
numbers = [1, 2, 3, 4, 5, 6]

# funkcja lambda sprawdzająca, czy liczba jest parzysta
is_even = lambda x: x % 2 == 0

# filtracja elementów z użyciem filterfalse()
filtered = itertools.filterfalse(is_even, numbers)

# wyświetlanie przefiltrowanych elementów
for item in filtered:
    print(item)
```

```python
import itertools

# lista zawierająca kody pocztowe
postal_codes = [
    '02-001', '02-002', '05-001', '05-002', '03-001', '03-002', '03-003'
]

# funkcja zwracająca pierwsze dwie cyfry kodu pocztowego
get_prefix = lambda code: code[:2]

# grupowanie kodów pocztowych według pierwszych dwóch cyfr
grouped = itertools.groupby(postal_codes, get_prefix)

# wyświetlanie grup i przynależnych do nich elementów
for prefix, group in grouped:
    print(f"Prefix: {prefix}")
    print(f"Codes: {list(group)}")
    print()
```

```python
import itertools

# lista zawierająca liczby od 0 do 9
numbers = list(range(10))

# wycinanie 5 elementów z listy od indeksu 2
sliced = itertools.islice(numbers, 2, 7)

# wyświetlanie wyciętych elementów
for item in sliced:
    print(item)
```

```python
import itertools

# lista zawierająca liczby od 1 do 5
numbers = [1, 2, 3, 4, 5]

# zwracanie kolejnych par elementów listy
pairs = itertools.pairwise(numbers)

# wyświetlanie kolejnych par
for pair in pairs:
    print(pair)
```

```python
import itertools

# funkcja zwracająca sumę dwóch liczb
def add(x, y):
    return x + y

# sekwencja krotek zawierających dwie liczby
numbers = [(1, 2), (3, 4), (5, 6)]

# wywołanie funkcji add() dla każdej krotki z użyciem starmap()
results = itertools.starmap(add, numbers)

# wyświetlanie wyników
for result in results:
    print(result)
```

```python
from itertools import takewhile

def mniejsze_niz_5(x):
    return x < 5

liczby = [1, 3, 7, 2, 4, 6, 8]

nowa_lista = list(takewhile(mniejsze_niz_5, liczby))

print(nowa_lista)
```

```python
import itertools

# Oryginalna sekwencja, dla której chcemy utworzyć niezależne iteratory
sequence = [1, 2, 3, 4, 5]

# Utworzenie 3 niezależnych iteratorów na podstawie oryginalnej sekwencji
iterators = itertools.tee(iter(sequence), 3)

# Wyświetlenie elementów z każdego iteratora
for i, iterator in enumerate(iterators):
    print(f"Iterator {i + 1}:")
    for item in iterator:
        print(item)
    print()
```

```python
from itertools import zip_longest

lista1 = [1, 2, 3]
lista2 = ['a', 'b']

for element in zip_longest(lista1, lista2, fillvalue=0):
    print(element)
```

```python
from itertools import product

liczby = [1, 2, 3]
litery = ['a', 'b']

kombinacje = list(product(liczby, litery))

print(kombinacje)
```

```python
import itertools

# Sekwencja, dla której chcemy wygenerować permutacje
sequence = ['A', 'B', 'C']

# Utworzenie iteratora, który generuje permutacje o długości 3 (domyślnie d
permutations_iterator = itertools.permutations(sequence, 3)

# Wyświetlenie elementów z iteratora
for item in permutations_iterator:
    print(item)
```

```python
import itertools

# Sekwencja, dla której chcemy wygenerować kombinacje
sequence = ['A', 'B', 'C', 'D']

# Utworzenie iteratora, który generuje kombinacje o długości 2
combinations_iterator = itertools.combinations(sequence, 2)

# Wyświetlenie elementów z iteratora
for item in combinations_iterator:
    print(item)
```

```python
from itertools import combinations_with_replacement

litery = ['a', 'b', 'c']

kombinacje = list(combinations_with_replacement(litery, 2))

print(kombinacje)
```

Bibliografia

- Dokumentacja języka Python.