Funkcje

Zadanie - proszę na dowolnej funkcji przetestować parametry pozycyjne, z nazwami, *args i **kwargs. Sprawdzić w jakiej kolejności mogą być ustawiane. Czy można bez błędu wywołać poniższą funkcję bez wprowadzania w niej zmian.

```
def person_print(name, last_name, *others, age):
    formatted_data = 'Imię: {}, nazwisko: {},
            wiek: {}'.format(name,last_name,age)
    others_str = ' '
    for arg in others:
        others_str += arg + ' '
    print(formatted_data + others_str)
```

Funkcje

Podobnie jak można zapakować parametry, można też rozpakować do parametrów.

Z listy:

```
def person_print(name, last_name, age):
    formatted_data = 'Imię: {}, nazwisko: {},
            wiek: {}'.format(name, last_name, age)
    print(formatted_data)


person_list = ['Witold', 'Bod', 19]
person_print(*person_list)  # Imię: Witold,
                            nazwisko: Bod, wiek: 19
```

Funkcje

Podobnie jak można zapakować parametry, można też rozpakować do parametrów.

Ze słownika:

```
def person_print(name, last_name, age):
    formatted_data = 'Imię: {}, nazwisko: {},
            wiek: {}'.format(name, last_name, age)
    print(formatted_data)


person_dict = {'age': 19, 'last_name': 'Bod',
                                'name': 'Witold'}
person_print(**person_dict)  # Imię: Witold,
                            nazwisko: Bod, wiek: 19
```

Asercje

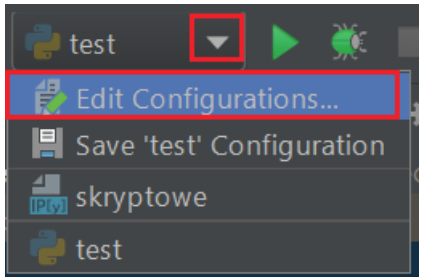
Asercje (assert) służą do umieszczania w kodzie warunków, które jeżeli nie zostaną spełnione wywołają komunikat o błędzie, co ułatwia debugowanie programu.

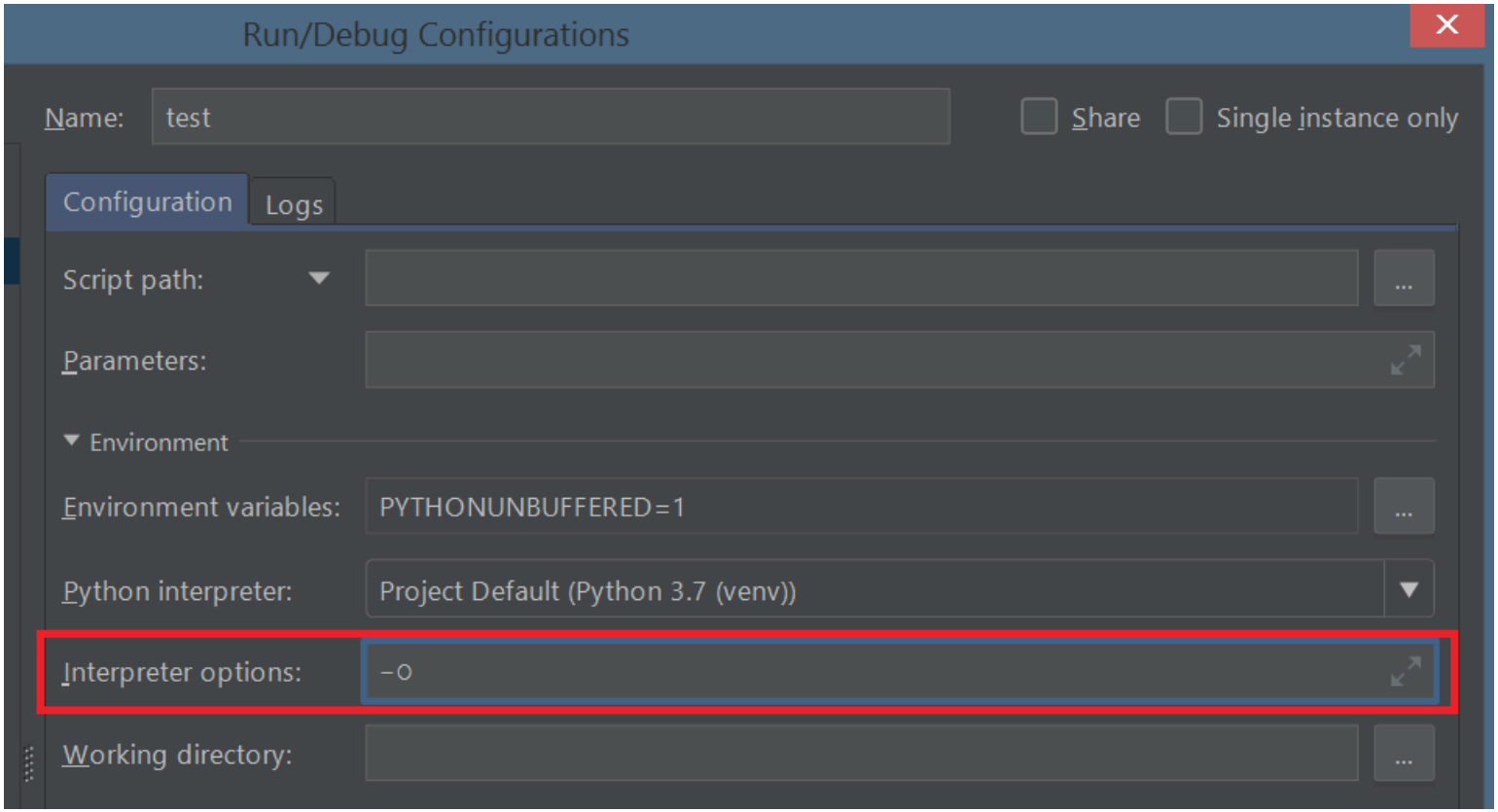
Wiadomość po przecinku jest opcjonalna.

```
def reduce_by(percent, value):
    assert 0 < percent <= 100, 'Nie można
                zredukować o {}%.'.format(percent)
    assert 0 < value, 'Wartość nie może być
                                mniejsza od 0.'
    return (100-percent)/100 * value
reduced_value = reduce_by(-10, 1200)
print(reduced_value)  # Bez asercji: 1320.0
```

Asercje

Zaletą asercji jest to, że można je wszystkie wyłączyć podając przy uruchamianiu programu opcję -O.





Pliki

Pliki można otworzyć za pomocą funkcji open( ) podając jako parametr względną lub bezwzględną ścieżkę do pliku w postaci napisu(string).

Obsługa pliku:
- Otwarcie pliku (zwraca uchwyt do pliku - filehandle).
- Użycie uchwytu, aby wykonać operację odczytu lub zapisu.
- Zamknięcie pliku.

```
txt_file = open('pliki/plik.txt')
print(txt_file.read())
txt_file.close()
# print(txt_file.read())  # I/O operation on
closed file.
print(txt_file.closed)  # True
```

Pliki

Próba odczytu z zamkniętego pliku zakończy się błędem.

```
# print(txt_file.read())  # I/O operation on
closed file.
```

Można sprawdzić, czy plik został zamknięty.

```
print(txt_file.closed)  # True
```

Pliki

Otwarty w taki sposób plik jest tylko do odczytu. Jeśli potrzebne są jakieś inne operacje to trzeba przekazać drugi parametr do funkcji open( ) określający tryb dostępu.

```
Plik otwarty do:
'r' odczytu
'w' zapisu (utworzy plik, jeśli nie istnieje)
'x' zapisu (FileExistsError jeśli plik już istnieje)
'a' dopisywania(append) zamiast nadpisywania

'b' tryb binarny (dodawany do innego trybu)
'+' dodany do innego trybu rozszerza go o czytanie
(w+) lub zapisywanie (r+). Ten ostatni nie kasuje
zawartości pliku w przeciwieństwie do w/w+.
```

Pliki

Drugi parametr funkcji open() można poszerzyć o:

'b' tryb binarny (dodawany do innego trybu)
'+' dodany do innego trybu rozszerza go o czytanie (w+) lub zapisywanie (r+). Ten ostatni nie kasuje zawartości pliku w przeciwieństwie do w/w+.

Pliki

Tryby dostępu.

```
# Zawartość w pliku zostanie nadpisana.
txt_file = open('pliki/plik.txt', 'w')
txt_file.writelines([str(number)+'\n' for number
                                                in
                                          (1, 200)])
txt_file.close()


# Zawartość w pliku zostanie nienaruszona i dodany
                                 będzie napis 'koniec'.
txt_file = open('pliki/plik.txt', 'r+')
print(txt_file.read())
txt_file.write('koniec')
txt_file.close()
```

Pliki

Metoda seek( ) - przesuwa aktualną pozycję od której następuje czytanie lub pisanie na numer podany w bajtach (znakach).
tell( ) - zwraca aktualną pozycję.

```
txt_file = open('pliki/plik.txt', 'r+')
print(txt_file.read())  # 0123456789
txt_file.seek(3)
txt_file.write('def')
txt_file.seek(0)
print(txt_file.tell())  # 0
print(txt_file.read(8))  # 012def67
txt_file.close()
```

Pliki

Zadanie: Proszę otworzyć plik i zapisać do niego dowolne dane, a następnie bez zamykania go spróbować odczytać jego zawartość.

Wyjątki

W momencie, kiedy następuje, na przykład, próba otwarcia nieistniejącego pliku lub zapisu do pliku będącego w trybie tylko do odczytu, zostanie wyrzucony wyjątek i program zakończy działanie.

```
txt_file = open('pliki/tylko_odczyt.txt', 'w')
txt_file.write('Linia tekstu.')
txt_file.close()
```

```
PermissionError: [Errno 13] Permission denied:
                                'pliki/tylko_odczyt.txt'
```

Wyjątki

Wyjątki, zwłaszcza te, na których pojawienie się nie mamy wpływu, należy obsłużyć. Służą do tego klauzule try, except. W bloku try umieszcza się kod, który może spowodować wyjątek, natomiast instrukcje bloku except są wykonywane w momencie, kiedy wystąpi wyjątek.

```
try:
    txt_file = open('pliki/tylko_odczyt.txt', 'w')
    txt_file.write('Linia tekstu.')
    txt_file.close()
except:
    print('Nastąpił wyjątek.')
```

Wyjątki

Wyjątki to tak naprawdę klasy (poza zakresem tego kursu). Wszystkie dziedziczą po klasie Exception. Są oczywiście różne typy wyjątków dopasowane do konkretnych sytuacji, jak na przykład OSError (wywoływany, gdy system operacyjny nie może wykonać zadania, takiego jak obsługa pliku) lub ZeroDivisionError (wywoływany, gdy drugim argumentem operacji dzielenia jest zero).

```
try:
    txt_file = open('pliki/tylko_odczyt.txt', 'w')
    txt_file.write('Linia tekstu.')
    txt_file.close()
except IOError as ioe:  # Doprecyzowanie typu
                                        wyjątku.
    print(ioe)  # Wypisanie informacji związanej z
                                    danym typem wyjątku.
```

Wyjątki

Można przechwycić więcej wyjątków niż jeden za pomocą rozbudowanej klauzuli except:

```
try:
    txt_file = open('pliki/plik.txt', 'w')
    division_result = int(input('dzielna: ')) /
                          int(input('dzielnik: '))
    txt_file.write(str(division_result))
    txt_file.close()
except (IOError, ZeroDivisionError) as e:
    print(e) #  division by zero
```

Wyjątki

… lub kolejnych klauzulach except:

```
try:
    txt_file = open('pliki/plik.txt', 'w')
    division_result = int(input('dzielna: ')) /
                          int(input('dzielnik: '))
    txt_file.write(str(division_result))
    txt_file.close()
except ZeroDivisionError as zde:
    print(zde)  # division by zero
except IOError as ioe:
    if txt_file:
        print(ioe)
```

Wyjątki

Pytanie: Jaki problem zaistnieje w momencie pojawienia się wyjątku `ZeroDivisionError`?

```
try:
    txt_file = open('pliki/plik.txt', 'w')
    txt_file.write('Wynik dzielenia: ')
    division_result = int(input('dzielna: ')) /
                          int(input('dzielnik: '))
    txt_file.write(str(division_result))
    txt_file.close()
except (IOError, ZeroDivisionError) as e:
    print(e) #  division by zero
```

Wyjątki

Pytanie: Jaki problem zaistnieje w momencie pojawienia się wyjątku `ZeroDivisionError`?

Plik pozostanie otwarty, a wcześniejsze instrukcje, które wpisywały coś do niego nie przyniosą oczekiwanego rezultatu.

```
try:
    txt_file = open('pliki/plik.txt', 'w')
    txt_file.write('Wynik dzielenia: ')
    division_result = int(input('dzielna: ')) /
                          int(input('dzielnik: '))
    txt_file.write(str(division_result))
    txt_file.close()
except (IOError, ZeroDivisionError) as e:
    print(e) #  division by zero
```

Wyjątki

Sprawdzenie, czy plik jest nadal otwarty:

```
txt_file = None
try:
    txt_file = open('pliki/plik.txt', 'w')
    txt_file.write('Wynik dzielenia: ')
    division_result = int(input('dzielna: ')) /
                          int(input('dzielnik: '))
    txt_file.write(str(division_result))
    txt_file.close()
except (IOError, ZeroDivisionError) as e:
    print(e)  # division by zero
    if txt_file:
        print(txt_file.closed)  # False
print(txt_file.closed)  # False
```

Wyjątki

Jest otwarty, czyli trzeba go zamknąć w momencie, kiedy wszystko pójdzie dobrze jak również w momencie pojawienia się wyjątku:

```
txt_file = None
try:
    txt_file = open('pliki/plik.txt', 'w')
    txt_file.write('Wynik dzielenia: ')
    division_result = int(input('dzielna: ')) /
                          int(input('dzielnik: '))
    txt_file.write(str(division_result))
    txt_file.close()
except (IOError, ZeroDivisionError) as e:
    print(e)  # division by zero
    if txt_file:
        txt_file.close()
```

Wyjątki

Zamiast robić to w dwóch miejscach można skorzystać z finally:

```
txt_file = None
try:
    txt_file = open('pliki/plik.txt', 'w')
    txt_file.write('Wynik dzielenia: ')
    division_result = int(input('dzielna: ')) /
                          int(input('dzielnik: '))
    txt_file.write(str(division_result))
except (IOError, ZeroDivisionError) as e:
    print(e)  # division by zero
finally:
    if txt_file:
        txt_file.close()
print(txt_file.closed)  # True
```

Wyjątki

Da się to zrobić jeszcze prościej z with:

```
try:
    with open('pliki/plik.txt', 'w') as txt_file:
        txt_file.write('Wynik dzielenia: ')
        division_result = int(input('dzielna:'))/
                          int(input('dzielnik:'))
        txt_file.write(str(division_result))
except (IOError, ZeroDivisionError) as e:
    print(e)  # division by zero
print(txt_file.closed)  # True
```

Wyjątki i funkcje

```
def division_saving():
        with open('pliki/plik.txt', 'w') as
                                        txt_file:
            txt_file.write('Wynik dzielenia: ')
            division_result = int(input('dzielna:'
                    )) / int(input('dzielnik: '))
            txt_file.write(str(division_result))


try:
    division_saving()
except (IOError, ZeroDivisionError) as e:
    print(e)  # division by zero
```

Wyjątki

Można samemu zgłaszać wyjątki za pomocą raise a nawet deklarować swoje:

```
class DoNotLikeOnesError(Exception):
    def __init__(self, msg):
        super().__init__(msg)
try:
    with open('pliki/plik.txt', 'w') as txt_file:
        …
        if division_result == 1:
            raise DoNotLikeOnesError('Nie lubię
                                      jedynek.')
        txt_file.write(str(division_result))
except (IOError, ZeroDivisionError,
                    DoNotLikeOnesError) as e:
    print(e)
```

Pliki

Zapis listy do pliku:

```
file_lines = ['Pierwsza linia', 'Druga linia',
   'Trzecia Linia', 'Czwarta linia', 'Piąta linia']
file_lines = [line+'\n' for line in file_lines]
try:
    with open('pliki/plik.txt', 'w') as txt_file:
        txt_file.writelines(file_lines)
except IOError as ioe:
    print(ioe)
```

Pliki i pętle

Odczytywanie po jednym znaku za pomocą pętli while:

```
with open('pliki/plik.txt', 'r') as txt_file:
    one_char = txt_file.read(1)
    while one_char:
        print(one_char, end='')
        one_char = txt_file.read(1)
```

Pliki i pętle

Odczytywanie po jednej linii za pomocą pętli for:

```
with open('pliki/plik.txt', 'r') as txt_file:
    for line in txt_file.readlines():
        print(line, end='')
```

Nie ma potrzeby używać readlines( ) - wystarczy sam uchwyt do pliku:

```
with open('pliki/plik.txt', 'r') as txt_file:
    for line in txt_file:
        print(line, end='')
```

Pliki i pętle

Zadanie – za pomocą with open( ) proszę odczytać zawartość jednego pliku i zapisać w drugim. Proszę uwzględnić obsługę wyjątków.