1. Operacje na plikach

Obok standardowego wejścia/wyjścia (input()/print()), równie przydatnym źródłem danych w programie mogą pliki. Aby rozpocząć pracę z plikami (w celu ich czytania albo pisania), należy najpierw użyć funkcji ***open****('nazwa pliku', tryb)*, która zwróci uchwyt (obiekt klasy *file*) na dany plik, a następnie za pomocą tak stworzonego uchwytu i metod takich jak ***read()***, ***readline()*** albo ***write('tekst'),*** odpowiednio czytać plik lub zapisywać do niego. Po zakończeniu wszystkich działań, plik należy zamknąć wywołując metodę ***close()***.

Przykład:

```
poem = '''
Programming is fun
When the work is done
if you wanna make your work also fun:
    use Python!
'''
# Otwarcie pliku w celu pisania(w)
f = open('poem.txt', 'w')
# Zapisanie tekstu zawartego w poem do pliku
f.write(poem)
# Zamknięcie pliku
f.close()
# Otwarcie pliku w celu czytania
# Jeśli tryb nie został podany domyślnie jest to 'r' czyli czytanie
f = open('poem.txt')
while True:
    line = f.readline()
    # Czytamy linia po linii dopóki nie będzie końca pliku
    # Długość wczytanej linii równa zero oznacza EOF - end of file
    if len(line) == 0:
        break
    print(line)
# Zamykamy plik
f.close()
```

Podsumowanie:

open('nazwa pliku', tryb) – otwiera plik i zwraca uchwyt do niego. Plik może być otworzony w celu: czytania(r), pisania(w) lub dopisywania(a). (Trybów jest w rzeczywistości więcej, ale nie będą tutaj omawiane).

read() – czyta cały plik i zwraca tekst w postaci stringa

readline() – czyta jedną linię pliku, przechodzi do następnej, zwraca wczytany tekst w postaci stringa

write('tekst') – zapisuje do pliku tekst podany jako argument

writelines(seq) – zapisuje do pliku sekwencję stringów (może to być np. lista stringów)

close() – zamyka plik

Podobnych metod jest znacznie więcej, ale powyższe metody są „najpotrzebniejsze” – zainteresowanych odsyłam do:

https://docs.python.org/3/tutorial/inputoutput.html

2. Wyjątki

Wyjątki służą w językach programowania do obsługi sytuacji… wyjątkowych ☺ Np.: Co jeśli chcemy otworzyć plik, którego nie ma na dysku? Programista piszący kod musi przewidzieć tego typu sytuację i ją oprogramować posługując się mechanizmem wyjątków.

Przykład:

Spróbuj podczas wykonywania następującego kodu:

```
s = input('Enter something --> ')
```

wcisnąć kombinację klawiszy: Ctrl + D. Jeśli wykonałeś polecenie poprawnie powinieneś dostać następujący komunikat na konsoli:

```
>>> s = input('Enter something --> ')
Enter something --> Traceback (most recent call last):
File "<stdin>", line 1, in <module>
EOFError
```

Python zgłosił błąd typu *EOFError*, który oznacza, że rozpoznał symbol końca pliku, którego tutaj nie oczekiwał.

Jak sobie z tym prawidłowo poradzić?

```
try:
    text = input('Enter something --> ')
except EOFError:
    print('Why did you do an EOF on me?')
except KeyboardInterrupt:
    print('You cancelled the operation.')
else:
    print('You entered {}'.format(text))
```

Output:

```
# Press ctrl + d
$ python exceptions_handle.py
Enter something --> Why did you do an EOF on me?

# Press ctrl + c
$ python exceptions_handle.py
Enter something --> ^CYou cancelled the operation.
```

```
$ python exceptions_handle.py
Enter something --> No exceptions
You entered No exceptions
```

Jak to działa?

Kod który może “rzucić” wyjątek (tak się na to mówi), umieszczamy w bloku ***try***, następnie umieszczamy „handlery”, konkretnych wyjątków (błędów). W przykładzie znajduję się także cześć *else* która zostanie wykonana jeśli żaden wyjątek nie zostanie „złapany” – nie będzie błędów. Tę część kodu możemy także umieścić bezpośrednio w bloku *try* razem z kodem, który potencjalnie może zgłosić wyjątek – wtedy już bez słowa kluczowego *else.*

Uwagi:

1. Jeśli po słowie kluczowym *except* nie ma nazwy wyjątku to złapany zostanie każdy wyjątek jaki ewentualnie zostanie rzucony.
2. W bloku try/except musi się znajdnować przynajmniej jeden except.

Programista może także sam w wybranym momencie rzucić wyjątek (zdefiniowany przez siebie lub jeden z wielu dostępnych). Służy do tego słowo kluczowe *rise*.

Przykład:

```
try:
    liczba = input('Podaj liczbę różną od 0')
    if liczba == 0:
        raise ValueError()
    print('Wpisałes {}'.format(liczba))
except ValueError:
    print('Czytaj ze zrozumieniem!')
```

Jak to działa?

W sytuacji gdy uznamy, że zdarzyła się sytuacja wyjątkowa, w tym przypadku użytkownik wpisał niepoprawną wartość, rzucamy wyjątek, który wyżej jest łapany i odpowiednio obsługiwany. Zauważ, że nie użyliśmy w tym przypadku *else* – kod, który chcemy żeby się wykonał umieszczony został bezpośrednio w *try* (ale po linii, w której zgłaszany jest wyjątek).

Ciekawostki dla zainteresowanych (do doczytania):

1. Programista może definiować własne wyjątki przez rozszerzanie klasy *Exception* (klasy poznamy w dalszym toku nauki)
2. Jeśli chcemy aby jakąś część kodu bezpośrednio związana z blokiem *try* wykonała się niezależnie od tego czy wystąpił wyjątek czy nie, możemy użyć słowa kluczowego *finally* lub *with.*