
	Politechnika Bydgoska im. J. J. Śniadeckich  <b>Wydział Telekomunikacji, Informatyki i Elektrotechniki</b>		
<b>Przedmiot</b>	Skryptowe języki programowania		
<b>Prowadzący</b>	mgr inż. Martyna Tarczewska		
<b>Temat</b>	<i>Wyjątki i pliki</i>		
<b>Student</b>			
<b>Nr ćw.</b>	6	<b>Data wykonania</b>	
<b>Ocena</b>		<b>Data oddania spr.</b>	

## 1. Cel ćwiczenia

Celem ćwiczenia jest poznanie sposobu korzystania z plików w języku Python oraz utrwalenie zdobytej wiedzy przez wykonanie zadań wymuszających obsługę plików i wyjątków. **Funkcje i zmienne stworzone podczas zajęć powinny być odpowiednio otypowane z użyciem biblioteki typing. Kod powinien być napisany w języku angielskim.**

## 2. Informacje podstawowe

### 2.1. Wyjątki

Niektóre działania w programach mogą spowodować pojawienie się błędów – może to być np. dzielenie przez 0 albo przejście przez zakres tablicy. Najczęściej jednak te błędy będą pojawiać się w kontakcie z użytkownikiem – użytkownik poda z konsoli niewłaściwe dane lub w przypadku korzystania z zewnętrznych źródeł (np. brak pliku, zerwane połączenie z bazą danych, zerwane połączenie TCP). Wyjątki *rzucamy* i *łapiemy*. W bloku *try* umieszczamy instrukcje, które potencjalnie mogą powodować pojawienie się błędów. Po słowie kluczowym *except* podajemy nazwy wyjątków, które chcemy łapać (które potencjalnie nasz kod może rzucić). Jak widać w jeden sposób można obsłużyć więcej niż jeden typ wyjątku. W poniższym kodzie kursywą zaznaczono sposób wypisania szczegółów wyjątku. Po konstrukcji *except* możemy dalej wykonywać bezpieczne instrukcje. Pełna lista wyjątków w Pythonie 3 znajduje się pod tym [linkiem](#).

```
input_string = "2,5"
try:
    some_number = 3/0
    some_number = float(input_string)
    some_number = float(input_string)
    print(f"This number is: {some_number}")
except (ValueError, UnicodeError) as ex1:
    print("cannot do it :(")
    print(ex1)
except NameError:
    print("i dont know this name :(")
except:
```

```
print("i dont know this error, sorry..")
```

Z wyjątkami wiążą się jeszcze trzy inne słowa kluczowe. Pierwszym jest *pass*, który można spotkać w bloku *except*. Należy jednak unikać jego stosowania, bo sprawia on, że kiedy wyjątek zostanie wyłapany, nic się nie stanie – będzie wyłapany, ale nie zostanie obsłużony.

Innym słowem jest *raise*, które spowoduje rzucenie konkretnego wyjątku. Poprawne są obie poniższe instrukcje.

```
x = -1
if x < 0:
    raise Exception("Sorry, no numbers below zero")

if x < 0:
    raise ValueError("Sorry, no numbers below zero")
```

Ostatnim elementem teoretycznym są bloki opcjonalne – blok *else* oraz blok *finally*. Blok *else* wykona się, jeśli wszystkie działania w bloku *try* zadziałały poprawnie (wyjątek nie został rzucony). Blok *finally* wykonuje się zawsze, bez względu na pojawienie się wyjątków lub ich brak.

```
try:
    print("Hello")
except:
    print("Something went wrong")
else:
    print("Everything went fine")
finally:
    print("I dont't care whether something went wrong")
```

## 2.2. Pliki tekstowe

W aplikacjach korzystamy nie tylko ze standardowego wejścia (konsola), ale też z innych źródeł danych, np. z plików. Sposób otwierania plików do odczytu pokazano poniżej. Należy zawsze podać nazwę pliku oraz tryb jego otwarcia „r”. Jeśli nie podamy ścieżki, to oznacza że plik jest w tym samym katalogu, co nasz plik wykonywalny. Podanie liczby spowoduje wyświetlenie początkowych znaków z pliku. Podano też konstrukcję do odczytywania pojedynczych linii, odczytania wszystkich linijek do listy oraz odczytu w pętli.

```
f = open("/sciezka/sciezka/plik.txt", "r")
print(f.read()) #odczyt całego pliku do końca
f = open("plik.txt", "r")
print(f.readline()) #odczyt jednej linii
f = open("plik.txt", "r")
print(f.read(10)) #odczyt pierwszych 10 znaków
f = open("thefile.txt", "r")
for x in f:
```

```
print(x) #odczyt w pętli
f = open("plik.txt", "r")
list = f.readlines() #wszystkie linie do listy
list = f.readlines(10) #10 pierwszych do listy
...
f.close() #otwarte pliki należy zamykać
```

Funkcja `open` może mieć jeszcze inne parametry: **a**, **w** oraz **x**. Ich działanie zostanie sprawdzone w jednym z zadań.

```
f = open("demofile.txt", "a")
f.write("Now the file has more content!")
f.close()

f = open("demofile.txt", "w")
f.write("Now the file has new content!")
f.close()

f = open("demofile.txt", "x")
f.write("Now the file has some content!")
f.close()
```

### 2.3. Zarządzanie plikami

W skryptach Pythona można w prosty sposób zarządzać plikami. W tym celu można zaimportować moduł `os` i/lub moduł `shutil`. Oczywiście są inne sposoby działania na plikach, jednak na potrzeby laboratorium można posłużyć się poniższymi instrukcjami.

```
import os
import shutil

#inny sposób odczytu pliku
with open('demofile.txt', 'r') as f:
    data = f.read()
    print(data)

#wypisanie listy plików i folderów
entries = os.listdir('exemplaryDir/')
for entry in entries:
    print(entry)

#wypisanie listy plików i folderów - wersja alternatywna
with os.scandir('exemplaryDir/') as entries:
    for entry in entries:
        if entry.is_file(): #dla plików
            print("file: " + entry.name)
        elif entry.is_dir(): #dla katalogów
            print("dir: " + entry.name)

os.mkdir('example_dir/') #stworzenie katalogu
```

```
os.makedirs('example_dir/10/05') #stworzenie kat. wraz z podkatalogami
os.remove('demofile.txt') #usunięcie pliku
os.rmdir('example_dir') #usunięcie katalogu
shutil.rmtree('example_dir') #usunięcie katalogu z zawartością
shutil.copytree('example_dir', 'backup_dir') #kopia katalogu
os.rename('oldName.txt', 'newName.txt') #zmiana nazwy
shutil.move('example_dir/', 'backup/') #przeniesienie pliku

src = 'path/to/file.txt'
dst = 'path/to/dest_dir'
shutil.copy(src, dst)
```

Zaznaczony kursywą fragment kodu to tzw. manager kontekstu. Po wyjściu z bloku *with* plik zostanie bezpiecznie zamknięty niezależnie od ewentualnych błędów. Jest to dość pomocne, gdyż bardzo często zapomina się o zamykaniu plików czy zwalnianiu innych zasobów. Poniżej pokazano jak za pomocą tego narzędzia przepisać zawartość z jednego pliku do drugiego.

```
with open("text.txt", "r") as file, open("text2.txt", "w") as file2:
    file2.write(file.read())
```

### 3. Przebieg ćwiczenia

#### 3.1. Zadanie 1.

Skopiować pierwszy fragment kodu z punktu 2.1. Wykonać go trzykrotnie – w każdym wywołaniu wybrać jeden sposób przypisania wartości zmiennej *some\_number*, pozostałe zakomentować. Jakie działanie powoduje wyłapanie którego wyjątku?

#### 3.2. Zadanie 2.

Skopiować drugi fragment kodu z punktu 2.1. Obudować fragmenty rzucające wyjątki blokiem *try* i obsłużyć pojawiające wyjątki w bloku *except*.

#### 3.3. Zadanie 3.

Wykorzystać trzeci fragment kodu z punktu 2.1. Zaproponować swoje działania, które może być umieszczone w bloku *try*, obsłużyć wyjątek w blokach *except*, *else* i *finally*.

#### 3.4. Zadanie 4.

Przetestować otwieranie plików (pierwszy fragment kodu z 2.2.). Jaki wyjątek zostanie rzucony w przypadku, gdy plik nie istnieje? Jaki wyjątek zostanie rzucony, jeśli będziemy chcieli zapisać do pliku? Zabezpieczyć otwieranie pliku odpowiednią konstrukcją.

#### 3.5. Zadanie 5.

Sprawdzić empirycznie działanie parametrów: *a*, *w* oraz *x* dla funkcji *open*. Spróbować wpisać zawartość (funkcja *write*) do utworzonego pliku oraz do nieistniejącego dla tych trzech parametrów. W sprawozdaniu dodać wypełnioną tabelkę.

Parametr funkcji open	Plik istnieje	Plik nie istnieje
r	Otwarcie pliku w trybie do pisania	Błąd ...
w		
a		
x		

### 3.6. Zadanie 6.

Napisać funkcję, która wyświetli drzewo katalogów dla wybranego katalogu.

### 3.7. Zadanie 7.

Sprawdzić działanie funkcji copy z modułu shutil. Wykonać kopiowanie na trzy sposoby:

- plik o danej nazwie nie istnieje w docelowym katalogu,
- plik o podanej nazwie istnieje w docelowym katalogu,
- katalog nie istnieje.

### 3.8. Zadanie 8.

Napisać funkcję przyjmującą trzy parametry: licznosc **n**, dolną granicę przedziału **a** oraz górną granicę przedziału **b**. Wylosować **n** liczb z zakresu  $\langle a, b \rangle$ . Zapisać te liczby w kolejnych liniach pliku tekstowego.

### 3.9. Zadanie 9.

Stworzyć dowolną klasę, której instancje posiadają własną listę, zmienną liczbową i napisową (może to być pracownik, samochód, student, figura geometryczna lub cokolwiek innego, można skorzystać z kodów napisanych na poprzednich zajęciach). Dla instancji stworzonej klasy umożliwić eksport aktualnego stanu instancji do pliku i jego przywrócenie z pliku. Zabezpieczyć program obsługą potencjalnych wyjątków.

### 3.10. Zadanie 10.

Utworzyć nową funkcję, która będzie działać na dwóch plikach jednocześnie. W pierwszym pliku zapisać kilka liczb (można wykorzystać zadanie 8). Do drugiego pliku przepisać te same liczby, jednak zapisać je w postaci binarnej.

## 4. Sprawozdanie

Sprawozdanie z laboratorium powinno zawierać:

- wypełnioną tabelę z początku instrukcji,
- kody programów będących rozwiązaniami wszystkich zadań wraz z komentarzami,
- demonstrację działania programu,
- odpowiedzi na pytania zawarte w zadaniach,

- wnioski.