Obsługa plików jest jedną z podstawowych umiejętności w przyborniku każdego programisty. Odczyt danych zapisanych w plikach, wczytywanie plików konfiguracyjnych, zapisywanie poszczególnych danych do plików – wszystko to wykorzystane w naszych programach wpłynie na ich użyteczność.

Zmienne plikowe

Do obsługi plików potrzebujesz specjalnej zmiennej, która będzie pełniła rolę **uchwytu do pliku.**

Wszelkie operacje związane z plikiem wykonuje się właśnie poprzez wykonywanie metod oferowanych przez zmienną plikową. Aby stworzyć zmienną plikową należy otworzyć plik.

Otwieranie pliku

Do otwierania pliku służy metoda **open**. Zwraca ona uchwyt do pliku. Metoda za argument przyjmuje **ścieżkę do pliku.**

```
f = open(filepath, "r")  # otwarcie pliku
```

Ścieżka do pliku

Jeżeli plik, który chcemy otworzyć znajduje się w tym samym katalogu co wykonywany plik Pythona wystarczy, że podamy jego nazwę (ścieżka **relative**).

Jeśli plik znajduje się w innym miejscu na naszym komputerze podajemy całą ścieżkę do pliku (ścieżka **absolute**).

```
filepath = "dane.txt"  # ścieżka relative
filepath = "E:\blog\dane.txt"  # ścieżka absolute
```

UWAGA! Jeżeli pracujesz na systemie Windows to podając pełną ścieżkę do pliku stosuj podwójne ukośniki (\) tak jak na powyższym kodzie. Jest to związane z tym, że ukośnik w napisach traktowany jest jako symbol ucieczki.

Jeśli podamy nazwę nieistniejącego pliku to zostanie on utworzony.

Tryby otwarcia pliku

Drugim często stosowanym argumentem metody jest tryb otwarcia pliku.

Najczęściej stosowane tryby to:

- "**r**" – plik otwarty do odczytu (read)
- "**w**" – plik otwarty do zapisu (write), przed zapisem zawartość pliku jest usuwana
- "**a**" – plik otwarty do zapisu, dodaje nową treść na końcu pliku, nie usuwa starej (append)

Domyślnie wszystkie pliki otwierane są w trybie **tekstowym**. Możemy otwierać je również w trybie **binarnym** dodając flagę '**b**'. Na przykład "rb" otworzy nam plik do oczytu binarnie.

Użycie operatora '+' pozwoli na otwarcie pliku zarówno do zapisu jak i odczytu (np "r+").

Mając już tę wiedzę możesz po prostu otworzyć plik

```
filepath = "E:\blog\dane.txt"
f = open(filepath, "r")
```

Kodowanie

Problemy z kodowaniem plików szczególnie często zdarzają się przy obsłudze polskojęzycznych tekstów.

Plik zapisany z nieprawidłowym kodowaniem może sprawić, że zamiast polskich znaków widać będzie dziwaczne symbole.

Aby naprawić ten problem często wystarczy otworzyć plik z kodowaniem w standardzie 'UTF-8'.

```
f = open(filepath, "r", encoding="utf-8")
f.read()
'żółć'
```

Otworzenie tego samego pliku z kodowaniem ASCII i próba czytania go wywoła błąd:

```
f = open(filepath, "r", encoding="ascii")
f.read()
Traceback (most recent call last):
  File "<pyshell#101>", line 1, in <module>
    f.read()
  File "C:Program Files (x86)Python36libencodingsascii.py", lin
e 26, in decode
    return codecs.ascii_decode(input, self.errors)[0]
UnicodeDecodeError: 'ascii' codec can't decode byte 0xc5 in
position 0: ordinal not in range(128)
```

Odczyt z pliku

Zawartość otwartego pliku możemy wczytać od razu w całości bądź czytać go linia po linii.

Odczyt całego pliku

Aby odczytać całą treść pliku skorzystaj z metody **read**.

```
f.read()
'Pierwsza linianDruga linianTrzecia linian123n3.14'
```

Za pomocą tej samej metody możemy przeczytać konkretną liczbę znaków podaną przez nas jako parametr metody:

```
f = open(filepath, "r")
f.read(8)
'Pierwsza'
```

UWAGA! Po odczytaniu zawartości pliku zostaje ona "**wykasowana"** ze zmiennej plikowej. W praktyce wygląda to tak, że "wskaźnik" odpowiadający za czytanie pliku przesuwa się. Można to porównać do przesuwania palcem po książce podczas czytania.

```
f = open(filepath, "r")
f.read(8)
'Pierwsza'
f.read()
' linianDruga linianTrzecia linian123n3.14'
```

Nie zdziw się więc, że po odczytaniu całego pliku następne wywołanie metody **read** zwróci pusty napis.

```
f = open(filepath, "r")
tekst = f.read()
f.read()
''
# teraz zobacz, że początkowo plik został odczytany
print(tekst[:14])
Pierwsza linia
```

Czytanie pliku liniami

Aby odczytać pojedyczną linię z pliku użyj polecenia **readline**.

```
 print(f.readline())
 f.readline()
'Pierwsza linian'
```

Jak widać zczytana została cała pierwsza linia pliku (wraz ze **znakiem końca linii**).

Ponowne użycie metody pozwoli na wczytanie następnej linii:

```
f.readline()
'Druga linian'
```

Plik możemy czytać **linia po linii** za pomocą pętli:

```
f = open(filepath, "r")
```

```
 for line in f:
    print(line)


Pierwsza linia

Druga linia

Trzecia linia

123

3.14
```

Aby pozbyć się dodatkowych odstępów możemy użyć parametru **end** w funkcji print:

```
f = open(filepath, "r")
 for line in f:
    print(line, end="")


Pierwsza linia
Druga linia
Trzecia linia
123
3.14
```

Inną opcją jest wczytanie wszystkich linii pliku do listy. Służy do tego metoda **readlines**.

```
f = open(filepath, "r")
 lines = f.readlines()
 lines
['Pierwsza linian', 'Druga linian', 'Trzecia linian', '123n',
'3.14']
```

Czytanie znak po znaku

Wcześniej pokazano, że można czytać konkretną liczbę znaków z pliku. Korzystając z tego możemy czytać plik **znak po znaku**:

```
f = open(filepath, "r")
 znak = f.read(1)
 while znak:
    print(znak)
    znak = f.read(1)

P
i
e
```

Zapis do pliku

Do zapisu do pliku służy metoda **write**.

```
f = open(filepath, "w")
 f.write("Dane")
4
 f.write("dane2")
5
```

UWAGA! Wartość podana do zapisu musi być **napisem**(stringiem)!

Do pliku zapisany został podany tekst. Jak widać na powyższym przykładzie metoda **write** zwraca liczbę zapisanych znaków.

Jeżeli chcesz, aby w pliku znalazł się znak nowej linii musisz dodać go samodzielnie.

```
 f = open(filepath, "w")
 f.write("Pierwsza linia\n")
15
 f.write("druga linia\n")
11
```

Do pliku możesz zapisać również wiele linii jednocześnie. Należy wykorzystać do tego listę napisów.

```
 f = open(filepath, "w")
 lines = ['pierwsza\n', 'druga\n', 'trzecia\n']
 f.writelines(lines)

 f = open(filepath, "w")
 print(f.read())
pierwsza
druga
trzecia
```

Jak widać powyżej, aby w pliku znalazły się znaki nowej linii należy dokleić je samodzielnie. Jeżeli jednak masz listę bez znaków nowej linii, możesz dodać je w ten sposób:

```
 lines = ['pierwsza', 'druga', 'trzecia']
 lines = [line + "\n" for line in lines]
 lines
['pierwsza\n', 'druga\n', 'trzecia\n']
```

Plik otworzony w trybie “w” przy otwarciu zostaje wyczyszczony. Aby dopisać tekst na końcu pliku użyj trybu “a”.

```
 f = open(filepath, "r")
 print(f.read())
Pierwsza linia
Druga linia

 f = open(filepath, "a")
 f.write("Trzecia linia\n")
14

 f = open(filepath, "r")
 print(f.read())
Pierwsza linia
Druga linia
Trzecia linia
```

Zamykanie pliku

Jeśli otworzysz plik to powinieneś również go zamknąć :) To dobra praktyka obsługi plików.

Służy do tego metoda **close**().

```
f = open(filepath, "r")
 f.close()
```

Co, jeśli nie zamknę pliku?

Otwarty w programie plik może powodować problemy. Inne aplikacje próbujące go odczytać nie będą mogły tego zrobić.

Jeżeli korzystasz z najpopularniejszej (i domyślnej) dystrybucji CPython to nawet jeśli sam nie zamkniesz pliku to przy końcu programu zostanie on zamknięty automatycznie.

Nie każda dystrybucja jednak robi to za nas. Dobrą praktyką jest zamykanie tego samodzielnie.

Istnieje jednak inny, bardziej Pythonowy sposób, niewymagający zamykania pliku.

Obsługa plików w Pythonowy sposób

Twórcy Pythona lubią ułatwiać sobie życie jak tylko mogą. Dlatego też wprowadzono alternatywną metodę obsługi plików, która nie wymaga dodatkowego zamykania pliku.

Alternatywna metoda wymaga wykorzystania struktury **with**:

```
with open(filepath, "w") as f:
    f.write("Zapisujemy do pliku\n")

20
 f.write("test")
Traceback (most recent call last):
```

```
  File "<pyshell#52>", line 1, in <module>
    f.write("test")
ValueError: I/O operation on closed file.
```

Jak widać po zakończonym bloku **with** plik został zamknięty.

Odczyt i zapis do 2 plików

Python pozwala w ten sposób otworzyć plik zarówno do zapisu, jak i odczytu:

```
with open('read_from.txt', 'r') as reader, open('write_to.txt',
'r') as writer:
    text = reader.read()
    writer.write(text)
```

Sprawdzanie pliku

Jak mogłeś zauważyć wyżej, próba pisania do zamkniętego pliku powoduje błąd.

Podobnie dzieje się przy próbie odczytu zamkniętego pliku, czy też zapisu do pliku nieprzeznaczonego do zapisywania.

Aby nie natrafić na błąd możesz skorzystać z jednej z metod służących do sprawdzania stanu pliku.

Aby sprawdzić czy plik został zamknięty sprawdź atrybut **closed()**.

```
 f = open(filepath, "r")
 print(f.closed)
False
 f.close()
 print(f.closed)
True
```

Do sprawdzenia czy plik otwarty jest do odczytu / zapisu służą metody **readable** / **writable**.

```
 f = open(filepath, "r")
 print(f.readable(), f.writable())
True False

 f = open(filepath, "w")
 print(f.readable(), f.writable())
False True

 f = open(filepath, "r+")
 print(f.readable(), f.writable())
True True
```