Pliki

- Pliki czyli nazwane pojemniki przechowujące dane na komputerze, którymi zarządza system operacyjny.
- Dostęp do plików z programów w Pythonie odbywa się za pomocą wbudowanej funkcji *open* tworzącej obiekt pliku Pythona
- Obiekt taki jest typem wbudowanym eksportującym metody służące do wykonywania zadań związanych z przetwarzaniem plików.
- Większość metod zajmuje się pobieraniem danych wejściowych i zapisywaniem danych wyjściowych
- Otwarcie pliku sprowadza się do wywołania funkcji open, przekazując jej nazwę pliku zewnętrznego i następnie trybu przetwarzania
- Tryby przetwarzania: **r** - (wartość domyślna) odczyt pliku, **w** - utworzenie pliku i otwarcie do zapisu, **a** - dodanie tekstu na końcu pliku, **b** - na końcu łańcucha trybu zezwala na dane binarne, **+** - oznacza otwarcie pliku zarówno do odczytu jak i zapisu

Pliki

- Przegląd metod wczytujących i zapisujących dane do pliku

Operacja	Interpretacja
`output = open(r'C:\spam', 'w')`	Utworzenie pliku do zapisu (`'w'` pochodzi od ang. *write* — zapis)
`input = open('data', 'r')`	Utworzenie pliku do odczytu (`'r'` pochodzi od ang. *read* — odczyt)
`input = open('data')`	To samo co w poprzednim wierszu (`'r'` jest trybem domyślnym)
`aString = input.read()`	Wczytanie całego pliku do jednego łańcucha znaków
`aString = input.read(N)`	Wczytanie następnych *N* bajtów (jednego lub więcej) do łańcucha znaków
`aString = input.readline()`	Wczytanie następnego wiersza (wraz ze znacznikiem końca wiersza) do łańcucha znaków
`aList = input.readlines()`	Wczytanie całego pliku do listy łańcuchów znaków z poszczególnymi wierszami
`output.write(aString)`	Zapisanie łańcucha bajtów do pliku
`output.writelines(aList)`	Zapisanie do pliku wszystkich łańcuchów znaków z wierszami znajdujących się w liście
`output.close()`	Ręczne zamknięcie (robione za nas, kiedy kończymy pracę z plikiem)
`output.flush()`	Opróżnienie bufora wyjściowego na dysk bez zamykania pliku
`anyFile.seek(N)`	Zmiana pozycji w pliku na wartość przesunięcia *N* dla następnej operacji
`for line in open('data'): użycie line`	Iteratory plików wczytują wiersz po wierszu
`open('f.txt', encoding='latin-1')`	Pliki tekstowe Unicode Pythona 3.0 (łańcuchy znaków `str`)
`open('f.bin', 'rb')`	Pliki bajtów binarnych Pythona 3.0 (łańcuchy znaków `bytes`)

Pliki

- Przykład demonstrujący podstawy przetwarzania plików

```
>>> myfile = open('myfile.txt','w')
>>> myfile.write('witaj, pliku tekstowy\n')
22
>>> myfile.write('żegnaj, pliku tekstowy\n')
23
>>> myfile.close()
>>> myfile = open('myfile.txt')
>>> myfile.readline()
'witaj, pliku tekstowy\n'
>>> myfile.readline()
'żegnaj, pliku tekstowy\n'
>>> myfile.readline()
''
```

- Jeżeli chcemy wyświetlić całą zawartość pliku, należy wczytać cały plik do jednego łańcucha znaków za pomocą metody *read*

```
>>> open('myfile.txt').read()
'witaj, pliku tekstowy\nżegnaj, pliku tekstowy\n'
>>> print(open('myfile.txt').read())
witaj, pliku tekstowy
żegnaj, pliku tekstowy
```

Pliki

- Jeśli natomiast chcemy przejrzeć plik tekstowy wiersz po wierszu, najlepszą opcją jest skorzystanie z *iteratorów plików*

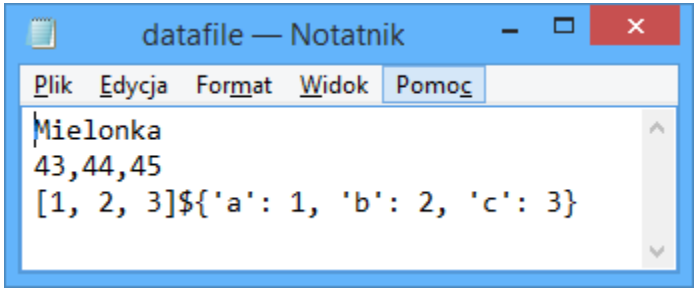
```
>>> for line in open('myfile.txt'):
        print(line, end = '')


witaj, pliku tekstowy
żegnaj, pliku tekstowy
```

- *Pliki tekstowe* reprezentują zawartość w postaci łańcuchów znaków str, wykonując automatycznie kodowanie i dekodowanie Unicode
- *Pliki binarne* reprezentują zawartość w postaci specjalnego typu łańcucha znaków o nazwie *bytes* i pozwalają programom na dostęp do niezmienionej zawartości plików. Nie można otworzyć pliku z danymi binarnymi w trybie tekstowym – dekodowanie Unicode się z dużym prawdopodobieństwem nie powiedzie

Pliki

- Przechowywanie obiektów Pythona w plikach tekstowych i przetwarzanie ich
- *Dane z plików są zawsze w skryptach łańcuchami znaków, natomiast metody zapisu nie formatują za nas obiektów na łańcuchy znaków*

```
X, Y, Z = 43, 44, 45
S = 'Mielonka'
D = {'a': 1, 'b' : 2, 'c' : 3}
L = [1, 2, 3]
F = open('datafile.txt','w')
F.write(S+'\n')
F.write('%s,%s,%s\n' %(X,Y,Z))
F.write(str(L) + '$' +str(D) + '\n')
F.close()
```



- *Otwarcie i wczytanie całej zawartości w jednej operacji*

```
text = open('datafile.txt').read()
print(text)
>>>
========== RESTART: L:/Python/skrypt_9.py ==
Mielonka
43,44,45
[1, 2, 3]${'a': 1, 'b': 2, 'c': 3}
```

Pliki

- Przekształcenie łańcuchów znaków na obiekty Pythona

```
F = open('datafile.txt')
line = F.readline()
print(line)
print(line.rstrip())
line=F.readline()
print(line)

parts = line.split(',')
print(parts)
int(parts[1])
numbers = [int(P) for P in parts]
print(numbers)

line =F.readline()
print(line)
parts = line.split('$')
print(parts)
eval(parts[0])
objects = [eval(P) for P in parts]
print(objects)
```

```
Mielonka

Mielonka
43,44,45

['43', '44', '45\n']
[43, 44, 45]
[1, 2, 3]${'a': 1, 'b': 2, 'c': 3}

['[1, 2, 3]', "{'a': 1, 'b': 2, 'c': 3}\n"]
[[1, 2, 3], {'a': 1, 'b': 2, 'c': 3}]
>>>
```

Pliki

- Metoda *pickle* jest narzędziem pozwalającym na przechowywanie prawie każdego obiektu Pythona bezpośrednio w pliku bez konieczności dokonywania „ręcznej” konwersji na łańcuch znaków. Jest ogólnym narzędziem do formatowania i przetwarzania danych

```
>>> D = {'a': 1, 'b': 2}
>>> F = open('datafile.pkl','wb')
>>> import pickle
>>> pickle.dump(D, F)
>>> F.close()
>>> F = open('datafile.pkl','rb')
>>> E = pickle.load(F)
>>> E
{'a': 1, 'b': 2}
```