pyladies

Siadamy co drugi rząd!

pyladies

4.12

oPYracje na plikach!

pyladies

Directed by: Jan Śliski

28.02.2018

pyladies

WIFI:
PUT-events-WiFi
Login: user_69211
Hasło: 9my3ci6kaZI

CO NAS DZISIAJ CZEKA

1. Powtórzymy materiał z poprzednich zajęć
1. Otworzymy plik
2. Coś w nim umieścimy
3. Potem go zamkniemy
4. Znowu otworzymy
5. Odczytamy jego zawartość ;)
6. Znowu coś zapiszemy
7. A potem zadanka!

POWTÓRKA

HTML - język znaczników, używany do opisu struktury strony internetowej.
Minimalny poprawny HTML (generowany przez PyCharm):

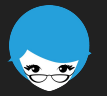
```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Title</title>
</head>
<body>

</body>
</html>
```

POWTÓRKA

To co znajduje się pomiędzy znacznikami `<head></head>` to informacje które nie są bezpośrednio widziane przez użytkownika (ale wykorzystywane przez przeglądarkę, lub też indeksery treści).

Pomiędzy znacznikami `<body></body>` umieszczamy zawartość, którą chcemy wyświetlić użytkownikowi.

Każdy tag może mieć pewne atrybuty definiujące dodatkowe właściwości, np:
`<p id="item-description"></p>` - atrybut `id` wskazuje na unikalny identyfikator dla danego tagu.
`<a href="http://google.com">Google</a>` - atrybut `href` wskazuje na adres przekierowania.



POWTÓRKA

Przykładowe znaczniki:

```
<div>
  <!-- Komentarz -->
  <!-- Tag div służy do grupowania innych znaczników -->
  <h1>Nagłówek pierwszego poziomu</h1>
  <p>Paragraf</p>
  <a href="http://google.com">Link</a>
  <img src="https://lumiere-a.akamaihd.net/v1/images/eu_xd_chi_pickle-and-peanut_r_5595fc24.jpeg" alt="Pickle">
  <h4 class="subsection-header">Nagłówek czwartego poziomu</h4>
  <ol>
     <li class="list-item list-first-item">A</li>
     <li class="list-item”>B</li>
     <li class="list-item>C</li>
  </ol>
</div>
```

POWTÓRKA

CSS - kaskadowe arkusze stylów, które definiują wygląd strony.
Dołączamy za pomocną tagu link, który umieszczamy najczęściej pomiędzy znacznikami head, np.

```
<head>
  ...
  <link href="style.css">
</head>
```

Style możemy tworzyć dla tagów, klas (nadawanym tagom za pomocą atrybutu `class`) oraz `id`-ków, lub ich kombinacji np.

```
body {...}                  # Selektor tagu
.ext-link {...}             # Selektor klasy
#item-desc {...}            # Selektor identyfikatora
p.klasa1.klasa2 {...}       # Selektor tagów <p> posiadających dwie klasy
li a {...}                  # Selektor tagów <a> będących dziećmi tagu <li>
```

POWTÓRKA - PRZYKŁAD CSS

```css
p {
  background-color: red;
  border: solid 2px yellow;
}

a {
  font-family: "Times New Roman";
}

a.external {
  font-family: "Arial";
}
```

Bardziej szczegółowe selektory mają pierwszeństwo nad tymi mniej szczegółowymi!

POWTÓRKA - PODSUMOWANIE

HTML, CSS i JS tworzą nierozerwalny fundament współczesnych stron internetowych. HTML odpowiada za strukturę, CSS za wygląd, a JS za interakcję pomiędzy stroną a użytkownikiem.

OPERACJE NA PLIKACH - SŁOWEM WSTĘPU

Ścieżka absolutna (bezwzględna):
C:\Users\jan.kowalski\Desktop\index.html - Windows, zaczynamy od litery dysku i dwukropka.
/home/jan.kowalski/desktop/index.html - Unix (Linux, MacOS), zaczynamy od “/”.

Ścieżka relatywna (względna) - ścieżka podawana w odniesieniu do folderu, w którym się znajdujemy (lub w którym znajduje się uruchomiony skrypt/otworzony plik HTML). Będąc w folderze jan.kowalski możemy odnieść się do pliku index.html w następujący sposób:
Desktop\index.html - Windows
desktop/index.html - Unix

OPERACJE NA PLIKACH - ZACZYNAMY!

Obiekty otwieramy za pomocą funkcji open() ...
Przyjmuje ona przynajmniej jeden argument - ścieżkę do pliku (relatywną lub absolutną) i zwraca obiekt pliku, np.

```
fl = open('F:\\pyladies\\navigation.html')
fl = open(r'F:\pyladies\navigation.html')

fl = open('/home/maciej/index.js')
fl = open('/home/maciej/cytaty.txt')
```

A zamykamy za pomocą metody .close()

```
fl.close()
```

Takie sprzątanie po sobie jest **bardzo**, **bardzo**, **bardzo** ważne!

OPERACJE NA PLIKACH

Dodatkowy parametr to tryb w jakim chcemy otworzyć plik:

```
fl = open('cytaty.txt', 'r')    # Otwórz w trybie odczytu (r -> read) (domyślny)
fl = open('cytaty.txt', 'w')    # Otwórz w trybie zapisu (w -> write) (nadpisuje)
fl = open('cytaty.txt', 'a')    # Otwórz w trybie zapisu (a -> append) (dopisuje)
fl = open('cytaty.txt', 'r+')   # Otwórz w trybie odczytu + zapisu (nadpisuje,
                                  ale nie usuwa całości)
fl = open('cytaty.txt', 'w+')   # Otwórz w trybie zapisu + odczytu + stwórz jeśli
                                  nie istnieje lub nadpisz istniejący
fl = open('cytaty.txt', 'b')    # Otwórz w trybie binarnym (b -> binary) - do
                                  otwierania wszystkiego co nie jest tekstem.
```

Aby zobaczyć wszystkie możliwości funkcji `open()` w PyCharm, po kliknięciu na funkcji wciśnij: Ctrl + B

OPERACJE NA PLIKACH - ODCZYT

Treść pliku po jego otwarciu możemy odczytać za pomocą metody: .read()

```
fl = open('cytaty.txt', encoding='UTF-8')
content = fl.read()          # Wczytaj całą zawartość pliku do zmiennej content
print(content)
fl.close()                   # WAŻNE!

fl = open('pasta.txt', encoding='UTF-8')
content = fl.read(14)        # Wczytaj pierwsze 12 znaków
print(content)               # ‘Kto szuka, ten’
fl.close()
```

encoding to dodatkowy parametr (przydatny, jeśli tekst ma polskie znaki)

OPERACJE NA PLIKACH - ODCZYT

```
fl = open('cytaty.txt', encoding="UTF-8")
content = fl.read(5)
print(content)                          # ???
content = fl.read(5)
print(content)                          # ???
fl.close()
```

Czytając plik kawałkami Python wie, gdzie skończył (ma taką zakładkę) i od tego momentu będzie wznawiał czytanie. Chyba, że skończy się plik.

OPERACJE NA PLIKACH - ZADANIE 1

Napisz skrypt, który:
- otworzy plik `message.txt` w trybie tylko do odczytu.
- w pętli będzie wczytywał po 50 znaków.
- jeśli skończył się plik - przerwij pętlę.
- jeśli we wczytanym stringu będzie ciąg znaków `'sekret'` to:
  weź z w/w stringa znaki na pozycjach: 1, 20, 45, 46 i 50.
  stwórz z nich nowy łańcuch znaków i zamień wszystkie znaki na wielkie litery, a następnie go wyświetl.
- zamknie plik.

Wskazówki:
Tablicę na string możemy zamienić za pomocą metody `''.join([])`
Jeśli plik się skończy, to metoda `.read()` zwróci pusty łańcuch znaków.
Pamiętaj, że indeksujemy od zera (pozycja 1, to indeks 0) ;)

OPERACJE NA PLIKACH - ODCZYT LINII

Aby odczytać jedną linię z pliku możemy użyć metody .readline()

```
fl = open('cytaty.txt', encoding="UTF-8")
print(fl.readline())
fl.close()

# ‘Kto szuka, ten najczęściej coś znajduje, niestety czasem zgoła nie
  to, czego mu potrzeba.’
```

Dodając opcjonalny parametr, możemy kontrolować ile pierwszych znaków z danej linii chcemy odczytać:

```
fl = open('cytaty.txt', encoding="UTF-8")
print(fl.readline(9))
fl.close()

# ‘Kto szuka’
```

OPERACJE NA PLIKACH - ZADANIE 2

Mamma mia! Ktoś miał przygotować spis postaci z powieści ‘Hobbit’, jednak plik z danymi wygląda jak najgorszy koszmar.
Ręcznie otwórz plik characters.txt i sprawdź, czy w danych występują jakieś prawidłowości, które możemy wykorzystać aby naprawić sytuację.
Jeśli tak jest, to napisz skrypt, który:

- otworzy plik characters.txt w trybie tylko do odczytu.
- wczyta pierwszą linię, a następnie:
  spróbuje do słownika przyporządkować znalezione wpisy, tak, aby kluczami były rasy, a wartościami - listy z imionami postaci.
- zamknie plik.

Wskazówki:
Łańcuch znaków możemy podzielić za pomocą metody .split() do której jako argument możemy przekazać znak, na podstawie którego string zostanie podzielony, np:

```
first_name, last_name = 'Jan-Kowalski'.split('-')
```

OPERACJE NA PLIKACH - ODCZYT WSZYSTKICH LINII

Aby odczytać wszystkie linie z pliku możemy użyć metody .readlines()

```
fl = open('cytaty.txt', encoding="UTF-8")
print(fl.readlines())
fl.close()
```

Należy jednak pamiętać, że przy dużych plikach wczytanie całej zawartości jest pomysłem złym i może się skończyć brakiem pamięci, a w konsekwencji powolnie działającym programem, który często się wywala :(

Jak sobie z tym radzić?

OPERACJE NA PLIKACH - ODCZYT WSZYSTKICH LINII

Aby odczytać wszystkie linie możemy użyć samego obiektu pliku!

```
fl = open('cytaty.txt', encoding='UTF-8')
for line in fl:
   print(line)
fl.close()
```

Takie podejście zagwarantuje nam, że Python nie wczyta do pamięci więcej niż musi i wszystko będzie śmigać!

OPERACJE NA PLIKACH - ZADANIE 3

Zbliża się okres składania PIT-ów i krasnoludzkie księgowe chciałyby rozliczyć członków wyprawy z zysków uzyskanych podczas wyprawy na Samotną Górę. Jakże przykrą niespodziankę sprawił im jednak Balin, który zamiast eleganckiego arkusza w Excelu dostarczył im naprędce sklejony plik tekstowy :(
Pomóż paniom księgowym ogarnąć ten bałagan i napisz skrypt, który:

- otworzy plik `members.txt`.
- stworzy słownik, w którym kluczami będą imiona bohaterów (pierwsza kolumna), a wartościami zagnieżdżone słowniki, zawierające pozostałe informacje (imię ojca, miejsce urodzenia, wiek, procentowy udział w zyskach).
- zamknie plik.

Wskazówki:
Brak wartości (‘-’) zastąp pustym łańcuchem znaków.

OPERACJE NA PLIKACH - Zapis do pliku

Zapisywanie do pliku odbywa się przy pomocy metody .write(), której jako parametr przekazujemy string, który chcemy zapisać w pliku. Wymagane jest też określenie odpowiedniego trybu w jakim plik ma zostać otwarty.

```
fl = open('praca-magisterska.txt', 'w+', encoding='UTF-8')
fl.write('Ala ma kota')
fl.close()
```

Należy zwrócić uwagę na tryb, w którym otwieramy plik do zapisu! Każdy tryb inny niż append spowoduje nadpisanie treści.

OPERACJE NA PLIKACH - ZADANIE 4

Kiedy kot wskoczy na klawiaturę - nie ma ratunku :(
Napisz skrypt, który:

- otworzy plik `cat.txt` w trybie do odczytu.
- stworzy nowy plik `fixed.txt` w trybie do zapisu (append).
- wczyta po kolei każdą linię z pliku `cat.txt` zamieniając wszystkie wystąpienia znaku '\' na '', po czym zapisze poprawioną linię w pliku `fixed.txt`.
- zamknie oba pliki.

Wskazówki:
Aby podmienić znaki w stringu użyj wbudowanej metody `.replace()`:

```
'Jan.Kowalski'.replace('.', ' ')
```

OPERACJE NA PLIKACH - ZADANIE 5

Jak wygląda pogoda na Śnieżce?
Napisz skrypt, który:

- stworzy nowy plik warunki-pogodowe.txt w trybie do zapisu (append).
- 30 razy, co sekundę wygeneruje:
  - czas - aktualną godzinę systemową
  - temperaturę - losową liczbę z zakresu <-35, -20>
  - ciśnienie - losową liczbę z zakresu <978, 1080>
  - wilgotność - losową liczbę z zakresu <50, 70>
  - prędkość wiatru - losową liczbę z zakresu <20, 60>
- zapisze w/w informacje w pliku warunki-pogodowe.txt w formacie: “YYYY-MM-DD HH:MM:SS, Temperatura: <wygenerowana_liczba>, Ciśnienie: <wygenerowana_liczba>,...” - całość w jednej linii.
- zamknie plik.

Wskazówki:
Na następnych slajdach.

OPERACJE NA PLIKACH - ZADANIE 5, WSKAZÓWKI

Generowanie liczb (pseudo)losowych:

```
import random                   # Umieszczamy na górze pliku
random.seed()                   # Inicjalizowanie generatora liczb pseudolosowych

print(random.randint(-30,-20))
print(random.randint(10, 100))
print(random.randint(-100,100))
```

Więcej o random:
https://docs.python.org/3/library/random.html

OPERACJE NA PLIKACH - ZADANIE 5, WSKAZÓWKI

Aby otrzymać aktualną datę systemową możemy użyć modułu datetime:

```
import datetime                    # Umieszczamy na górze pliku

print(datetime.datetime.now())     # 2018-02-27 01:39:14.222039
```

Jest prawie idealnie! Jak się pozbyć milisekund? Np. za pomocą metody .strftime(), której jako argument przekazujemy string z pożądanym formatowaniem.

```
import datetime

print(datetime.datetime.now().strftime("%Y-%m-%d %H:%M:%S"))
```

Więcej o datetime i o tym jak korzystać z pojedynczych pól oraz formatowania:
https://docs.python.org/3/library/datetime.html

OPERACJE NA PLIKACH - ZADANIE 5, WSKAZÓWKI

Jak zatrzymać wykonywanie programu Pythonie? Chociażby za pomocą funkcji sleep z modułu time:

```
import time

print("Hello!")
time.sleep(2)
print("Hello after 2 seconds!")
```

Więcej module time:
https://docs.python.org/3/library/time.html

OPERACJE NA PLIKACH - ZADANIE 5, WSKAZÓWKI

Formatowanie stringów można wykonać na kilka sposobów:

```
name = 'John'
last_name = 'Smith'

print(f'His name is {name} {last_name}')
```

Dostępny od Pythona w wersji 3.6, i jest to sposób preferowany!
Wymaga, żeby zmienne, których użyjemy w nawiasach klamrowych były zdefiniowane!
Alternatywny sposób, nie wymagający zdefiniowania zmiennych przed wstawieniem ich do szablonu:

```
template = 'His name is {0} {1}. {0} is happy!'
name = 'John'
last_name = 'Smith'

print(template.format(name, last_name))
```

OPERACJE NA PLIKACH - FORMAT JSON

Istnieje mnóstwo formatów i sposobów na zapis danych tekstowych, tak dużo, że aż samo przychodzi do głowy pytanie - którego używać? Jednym z najczęściej wykorzystywanych formatów przy przetwarzaniu danych (zwłaszcza w aplikacjach webowych) jest format JSON (JavaScript Object Notation). Cechuje go prostota w odczycie i zapisie danych zarówno przez człowieka, jak i przez maszyny oraz wbudowane wsparcie w wielu językach programowania.

Czas poznać JSON-a!

JSON VS SŁOWNIK

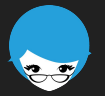
```
"""{
   "John Smith": {
      "occupation": "Software Engineer",
      "age": 30,
      "city": "London",
      "cars": [
         "Ford",
         "BMW"
      ]
   },
   "Mary Jane": {
      "occupation": "Lawyer",
      "age": 25,
      "city": "Bristol",
      "cars": [
         "Mercedes"
      ]
   }
}"""
```

```
{
   "John Smith": {
      "occupation": "Software Engineer",
      "age": 30,
      "city": "London",
      "cars": [
         "Ford",
         "BMW"
      ]
   },
   "Mary Jane": {
      "occupation": "Lawyer",
      "age": 25,
      "city": "Bristol",
      "cars": [
         "Mercedes"
      ]
   }
}
```

JSON VS SŁOWNIK

```
"""[
  {
    "name": "John Smith",
    "occupation": "Software Engineer",
    "age": 30,
    "cars": [
      "Ford",
      "BMW"
    ]
  },
  {
    "name": "Mary Jane",
    "occupation": "Lawyer",
    "age": 25,
    "cars": [
      "Mercedes"
    ]
  }
]"""
```

```
[
  {
    "name": "John Smith",
    "occupation": "Software Engineer",
    "age": 30,
    "city": "London",
    "cars": [
      "Ford",
      "BMW"
    ]
  },
  {
    "name": "Mary Jane",
    "occupation": "Lawyer",
    "age": 25,
    "city": "Bristol",
    "cars": [
      "Mercedes"
    ]
  }
]
```



CZY JSON == SŁOWNIK/LISTA?

Nie!

JSON to przede wszystkim sposób zapisu danych tekstowych. Słowniki i listy oraz wszystkie inne struktury w Pythonie, których reprezentacja tekstowa wygląda podobnie do JSON-a, to dużo bardziej skomplikowane obiekty - struktury danych, które umożliwiają dużo, dużo więcej!

Dane przechowywany w tych strukturach mogą jednak zostać bardzo łatwo zapisane w formacie JSON oraz z niego odczytane, co nazywamy odpowiednio: serializacją i deserializacją.

OBIEKT W PYTHONIE -> JSON - SERIALIZACJA

Aby obiekt języka Python na JSON możemy wykorzystać moduł json i metodę .dumps()

```
import json

a_dict = {
  "name": "Johnny",
  "last_name": "Walker"
}

a_json = json.dumps(a_dict)
print(a_json)                 # '{"name": "Johnny", "last_name": "Walker"}'
                              # W sensie: łańcuch znaków ;)
```

OBIEKT W PYTHONIE -> JSON - SERIALIZACJA DO PLIKU

Aby obiekt języka Python na JSON możemy wykorzystać moduł json i metodę .dump()

```
import json

a_dict = {
   "name": "Johnny",
   "last_name": "Walker"
}
fp = open('json-data.json', 'w')

a_json = json.dump(a_dict, fp)

fp.close()
```

JSON -> OBIEKT W PYTHONIE - DESERIALIZACJA

Aby zamienić JSON-a wczytywanego z pliku na obiekt języka Python możemy wykorzystać moduł json i metodę .loads()

```
import json

json_data = """{
  "name": "John Smith",
  "occupation": "Software Engineer",
  "age": 30,
  "city": "London",
  "cars": [
    "Ford",
    "BMW"
  ]
}"""


obj = json.loads(json_data)
print(obj['name'])          # ‘John Smith’
```

JSON -> OBIEKT W PYTHONIE (Z PLIKU)

Aby zamienić JSON-a wczytywanego z pliku na obiekt języka Python możemy wykorzystać moduł json i metodę .load() (bez ‘s’)

```
import json

fp = open('sample.json')

obj = json.load(fp)

fp.close()

print(obj['name'])          # ‘John Smith’
```

ZADANIE 6 - RICK AND MORTY

Stwórz skrypt, który:

- pobierze za pomocą API listę bohaterów z serwisu rickandmortyapi.com
- otrzymany JSON przekonwertuje na słownik
- stworzy nowy listę słowników, w której każdy słownik będzie odpowiadał jednej postaci i będzie zawierał informacje o imieniu, gatunku, płci oraz obrazku.
- otrzymaną listę zserializuje do formatu JSON, a następnie zapisze w pliku heroes.json

Wskazówki:
Skąd i jak dostać dane? Następny slajd ;)
JSON -> obiekt? Metoda .loads(), slajd 36
https://rickandmortyapi.com/api/character/ - adres z którego mamy pobrać dane

Nie masz internetu? Skorzystaj z pliku data.json z zadania 7!

ZADANIE 6 - WSKAZÓWKI

Aby pobrać dane z jakiegoś serwisu możemy wykorzystać moduł requests:

```
import requests

url = 'https://rickandmortyapi.com/api/character/' # Adres, który zwróci dane JSON-owe
response = requests.get(url)                       # Metoda .get() wysyła żądanie oraz
                                                     zwraca obiekt odpowiedzi,
                                                     posiadający szereg właściwości
if response.status_code == 200:                    # status_code == 200 mówi nam, że
                                                     serwer odebrał nasze żądanie i
                                                     poprawnie zwrócił odpowiedź
    heroes = response.json().get('results')        # metoda json() zwróci ciało
                                                     odpowiedzi jako obiekt pythonowy
                                                     (czyli robi takie json.loads() na
                                                     treści odpowiedzi)
```

ZADANIE 7 - ULTIMATE RICK AND MORTY COMBO

Skopiuj skrypt z poprzedniego zadania, a następnie zmodyfikuj go tak, że:

- pobierze za pomocą API listę bohaterów z serwisu rickandmortyapi.com
- otrzymany JSON przekonwertuje na słownik
- wczyta plik hero-template.html do zmiennej i dla każdego bohatera za pomocą formatowania stringów wstawi w odpowiednie miejsce: imię, rasę, oraz link do obrazka, oraz nazwę odpowiedniej klasy w zależności od tego, czy dany bohater żyje, czy nie.
- tak skonstruowany string doda do listy przygotowanych szablonów.
- połączy tak przygotowane stringi w całość po czym wczyta plik index-template.html do zmiennej i za pomocą formatowania stringów wstawi w/w połaczony string w odpowiednie miejsce, a całość zapisze do pliku index.html

Brzmi ciężko? Obczaj następne slajdy ;)

ZADANIE 7 - ULTIMATE RICK AND MORTY COMBO

Jak sformatować stringa? Bardzo prosto! Metoda .format() przychodzi nam z pomocą:

```
template_str = 'Hello! My name is {name}. I come from {country}.'
my_name = 'Borat'
my_country = 'Kazakhstan'

formatted_str = template_str.format(name=my_name, country=my_country)
print(formatted_str)
```

Alternatywnie (ale mniej czytelnie):

```
template_str = 'Hello! My name is {0}. I come from {1}.'
my_name = 'Borat'
my_country = 'Kazakhstan'

formatted_str = template_str.format(my_name, my_country)
print(formatted_str)
```

ZADANIE 7 - ULTIMATE RICK AND MORTY COMBO - WSKAZÓWKI

Nie masz dostępu do internetu? W pliku data.json znajduje się wynik zapytania pod wskazany URL, gotowy do wczytania przez json.load() ;)

Możesz też go wykorzystać, żeby zobaczyć jakich kluczy i wartości spodziewać się w zwracanym przez API JSON-ie!

(PS zrób tak!)

KĄCIK WIEDZY TAJEMNEJ

Pamiętacie to okropne dbanie o zamykanie pliku? Nie tylko wam przeszkadzało, więc ktoś wymyśli lepszy sposób - with:

```
f = open('plik.txt')
f.read()
f.close()
```

```
with open('plik.txt') as f:
  f.read()
```

Powyższe zapisy są równoznaczne, z tym, że w drugim przypadku oszczędzamy 1 linijkę kodu (!!!) oraz sporo nerwów przy debugowaniu ;)

```
with open('plik.txt') as plik1,\
   open('plik2.txt', 'w') as plik2:
  plik2.write(plik1.read())
```

Możemy w ten sposób otworzyć więcej niż jeden plik! ;)


QUARTERS ONLY
25¢
1. DROP COIN IN SLOT.
2. PULL HANDLE DOWN ALL THE WAY-LET GO
3. TO CLEAR VISION - TURN RED KNOB.
2414

Q&A