2 października 2015

Kurs języka Python

Lista 1.

Zadanie 1. Zaprogramuj funkcję `rzut_kostka()`, która symuluje rzut sześcienną kością do gry, tzn. funkcja ma zwracać losową liczbę naturalną z przedziału $[1 \dots 6]$. Korzystając z tej funkcji zaprogramuj prostą grę (z dwoma zawodnikami), w której w każdej turze każdy zawodnik rzuca dwie kostki (tj. dwukrotnie wywołuje funkcję rzut_kostka()). Turę wygrywa ten, kto wyrzuci więcej kostek. Grę wygrywa ten, kto po n turach ma więcej zwycięstw. Jeśli po n turach jest remis, gra toczy się do pierwszego zwycięstwa. Komputer ma grać sam ze sobą, a dodatkowo po każdej turze wypisywać na wylosowaną liczbę oczek oraz aktualny bilans zwycięstw i porażek. Liczba n jest parametrem wywołania programu.

Zadanie 2. Napisz program który szyfruje tekst za pomocą następującego algorytmu opartego na algorytmie XOR: do zaszyfrowania jest potrzebny klucz k, tj. liczba z przedziału $[0 \dots 255]$. Kolejne litery tekstu zamieniamy na odpowiedni kod ASCII, obliczamy wynik operacji XOR z k i do szyfrogramu wstawiamy wynik operacji zamieniony na odpowiedni znak ASCII. Na przykład tekst *Python* za pomocą klucza 7 (binarnie: 0000 0111) szyfrujemy tak:

litery	P	y	t	h	o	n
ASCII	0101 0000	0111 1001	0111 0100	0110 1000	0110 1111	0110 1110
XOR	0101 0111	0111 1110	0111 0011	0110 1111	0110 1000	0110 1001
szyfr	W	~	s	o	h	i

Program ma mieć postać funkcji `zaszyfruj(tekst, klucz)`, która dla podanego tekstu i klucza zwraca zaszyfrowany tekst. Zaprogramuj również funkcję `odszyfruj(szyfr, klucz)`.

Zadanie 3. Zaprogramuj funkcję `slownie(n)`, która zwraca string będący słownym zapisem liczby naturalnej n, zapisanej zgodnie z regułami języka polskiego. Można przyjąć, że argumentem jest liczba o co najwyżej 8 cyfrach.

Zadanie 4. Napisz jednoargumentową funkcję `rozklad(n)` która oblicza rozkład liczby n na czynniki pierwsze i zwraca jako wynik listę par $[(p_1, w_1), (p_2, w_2), \dots, (p_k, w_k)]$ taką, że
$n = p_1^{w_1} * p_2^{w_2} * \dots * p_k^{w_k}$ oraz $p_1, \dots, p_k$ są różnymi liczbami pierwszymi. Na przykład

```
>>> rozklad(756)
[(2, 2), (3, 3), (7, 1)]
```

Zadanie 5. Zaprogramuj funkcję `tabliczka(x1, x2, y1, y2)`, która wypisze na ekran tabliczkę mnożenia dla liczb $[x_1, \dots, x_2] \times [y_1, \dots, y_2]$; np.

```
>>> tabliczka(3,5, 2, 4)
  3  4  5
2 6  8  10
3 9  12 15
4 12 16 20
```

Możesz zaniedbać formatowanie (wyrównywanie kolumn).

Każde zadanie jest warte 2 punkty. Na pracowni do oceny należy przedstawić dwa zadania.

Marcin Młotkowski

9 października 2015

Kurs języka Python

Lista 2.

Zadanie 1. Zaprogramuj klasę ***HtmlObject*** jako podklasę klasy ***object***, która implementuje dwie metody:

- `html()`: metoda ta powinna zwracać string zawierający informacje o stanie obiektu, tj. lista pól wraz z aktualnymi typami tych pól i wartościami. String powinien być w formacie html (wystarczy prosty wariant html);
- `html_page()`: metoda ta powinna zwrócić pełną stronę html (z koniecznymi znacznikami) wraz ze stanem obiektu oraz ze stanem wszystkich obiektów wskazywanych przez pola obiektu, o ile dany obiekt implementuje metodę `html()`.

Zaprogramuj jakąś prostą podklasę ***HtmlObject*** i sprawdź działanie metody `html()`.

Zadanie 2. Zaprogramuj klasę ***ObiektyDodawalne***, która implementuje operator '+'. Wynikiem działania

```
>>> obj1 + obj2
```

gdzie `obj1` i `obj2` są obiektami klasy (a jeszcze lepiej podklasy!) ***ObiektyDodawalne*** a wynik obiektem klasy ***ObiektyDodawalne*** zawierający wszystkie pola `obj1` i `obj2`. W przypadku konfliktu, tj. gdy `obj1` i `obj2` zawierają pole o tej samej nazwie program powinien wybrać dowolną wartość i wypisać ostrzeżenie. Wykonaj to zadanie operując wprost na słowniku zmiennych obiektu. Zaimplementuj kontrolę poprawności danych, tj. sprawdzenie, czy `obj2` jest klasy (lub podklasy) ***ObiektyDodawalne***.

Zadanie 3. Wyjątki mogą służyć nie tylko do sygnalizacji jakiejś niepożądanej sytuacji, ale także do zakończenia działania funkcji i przekazania obliczonej wartości. Na przykład funkcja obliczająca rekurencyjnie silnię w końcowym etapie obliczeń wymaga wielokrotnego wykonania instrukcji `return wyrażenie` aby zwinąć stos wywołań rekurencyjnych. Zamiast tego wystarczy na końcu obliczeń zgłosić wyjątek `raise wynik` zawierający wynik obliczeń i w odpowiednim miejscu obsłużyć ten wyjątek. Zaprogramuj wersję rekurencyjną i z wywołaniem wyjątku funkcji obliczających

- silnię liczby n;
- n-ty element ciągu Fibonacciego.

Zbadaj (na przykład za pomocą modułu `timeit`) która wersja jest szybsza.

Każde zadanie jest warte 2 punkty. Na pracowni do oceny należy przedstawić dwa zadania.

Marcin Młotkowski

16 października 2015

Kurs języka Python

Lista 3.

Poniżej są zadania polegające na implementacji funkcji zwracających listy liczb naturalnych. Każde z zadań należy wykonać w dwóch wersjach: w wersji z listą składaną i wersję funkcyjną.

Wersja z *listą składaną* powinna być w postaci jednej listy składanej, zawierającej być może inną listę składaną. W przypadku bardzo długich wyrażeń akceptowane będzie wydzielenie podlisty składanej

```
def zadana_funkcja(n):
    lista_tymcz = [ lista skladana ]
    return [ lista_składana_zawierająca lista_tymcz ]
```

Implementacja funkcyjna powinna korzystać z funkcji dedykowanych do operacji na listach: `filter`, `map` czy `reduce`.

Zbadaj, która wersja jest szybsza.

Do zaprogramowania powyższych zadań wystarczą standardowe funkcje i operatory, nie ma potrzeby korzystania z dodatkowych modułów.

Zadanie 1. Zaprogramuj jednoargumentowe funkcje `pierwsze_skladana(n)` i `pierwsze_funkcyjna(n)`, które zwracają listę liczb pierwszych nie większych niż n, na przykład

```
>>> pierwsze(20)
[2, 3, 5, 7, 11, 13, 17, 19]
```

Zadanie 2. Zaprogramuj jednoargumentowe funkcje `doskonale_skladana(n)` i `doskonale_funkcyjna(n)`, które zwracają listę liczb doskonałych nie większych niż n, na przykład

```
>>> doskonale(1000)
[6, 28, 496, 8128]
```

Zadanie 3. Zaprogramuj jednoargumentowe funkcję `rozklad_skladana(n)` i `rozklad_funkcyjna(n)` które obliczają rozkład liczby n na czynniki pierwsze i zwracają jako wynik listę par $[(p_1, w_1), (p_2, w_2), \ldots, (p_k, w_k)]$ taką, że $n = p_1^{w_1} * p_2^{w_2} * \ldots * p_k^{w_k}$ oraz $p_1, \ldots, p_k$ są różnymi liczbami pierwszymi. Na przykład

```
>>> rozklad(756)
[(2, 2), (3, 3), (7, 1)]
```

Ponieważ w tym zadaniu może być potrzebna lista liczb pierwszych, można zaimplementować pomocniczą funkcję sprawdzającą pierwszość liczby bądź zwracającą listę liczb pierwszych. W przypadku tej funkcji pomocniczej implementacja może być dowolna.

Zadanie 4. Zaprogramuj jednoargumentowe funkcje `zaprzyjaznione_skladana(n)` i `zaprzyjaznione_funkcyjna(n)`, które zwracają listę par liczb zaprzyjaźnionych nie większych niż n, na przykład

```
>>> zaprzyjaznione(1300)
[(220, 284), (1184, 1210)]
```

Odpowiednie definicje można znaleźć np. w polskiej Wikipedii. Wybierz dwa z podanych zadań. Każde zadanie jest warte 2 pkt.

Marcin Młotkowski

23 października 2015

Kurs języka Python

Lista 4.

Zadanie 1. Zaimplementuj to samo zadanie które wykonałaś/wykonałeś z listy 3., ale wykorzystaj do tego iteratory. Zbadaj, która teraz wersja jest najszybsza.

Przetestuj działanie implementacji dla różnych argumentów, np. dla 10, 100, 1000 etc. Wypisz na konsolę czasy działania dla poszczególnych danych i implementacji w ładnie sformatowany sposób, na przykład

```
      | funkcyjna | skladana | iterator
   10 |     0.001 |    0.002 |   0.003
  100 |     0.010 |    0.020 |   0.030
 1000 |     0.100 |    0.200 |   0.300
```

Można też plik sformatować tak, aby wynik mógł być wejściem dla jakiegoś programu rysującego wykresy, np. *gnuplot* czy arkusz kalkulacyjny.

Zadanie 2. Zaprogramuj jako iterator kolekcję do przetwarzania plików tekstowych w formie 'akapit po akapicie'. Na przykład aby można było łatwo sformatować plik do postaci html'a:

```
for akapit in kolekcja:
  print('<p>' + akapit + '</p>')
```

`Kolekcja` powinna korzystać z dowolnego strumienia danych.

Zaprogramuj funkcję `formatuj_akapit(akapit, szerokość)` która zwróci ten sam tekst, jednak żaden wiersz nie będzie dłuższy niż `szerokość` (o ile będzie to możliwe). Jako prezentację działania programu utwórz kolekcję sformatowanych akapitów, np. korzystając z `map` (w wersji Pythona 3.*) lub analogicznej funkcji z pakietu `itertools`.

Zadanie 3. Zaprogramuj iterator, który przetwarza strumień tekstowy i zwraca kolejne zdania (stringi) z tego strumienia. Zaprogramuj funkcję `korekta(zdanie)` która sprawdza, czy zdanie zaczyna się wielką literą i kończy się kropką, oraz czy po kropce jest jakiś biały znak, a jako wynik zwraca skorygowane zdanie.

Jako prezentację działania programu utwórz kolekcję skorygowanych zdań, np. korzystając z `map` (w wersji 3.*) lub analogicznej funkcji z pakietu `itertools`.

Zadanie 4. Zaprogramuj iterator który przetwarza strumień tekstowy i zwraca kolejne słowa z tekstu (dla utrudnienia uwzględnij dzielenie słów na końcach wierszy), pomijając białe znaki i znaki interpunkcyjne. Korzystając z tej implementacji zaprogramuj obliczanie statystyki długości słów w tekście, tj. ile jest słów długości 1, ile długości 2 etc.

Na kolejną pracownię zaprogramuj zadanie 1. oraz jedno z zadań 2–4. Zadanie 1. jest warte 2 pkt., zadania 2–4: 3 pkt. Do zdobycia jest więc 5 pkt.

Marcin Młotkowski

30 października 2015

Kurs języka Python

Lista 5.

Zadanie 1. Zaprogramuj moduł służący do przeglądania stron WWW i wykonujący dla każdej strony pewną akcję. Akcja powinna być funkcją, przekazywaną jako parametr do modułu, a której argumentem jest strona html. Przyjmij, że przeglądanie rozpoczyna się od strony, której adres jest podany jako argument wywołania. Następnie przeszukiwane są te strony, do których linki znajdują się na bieżącej stronie. Zadbaj o to, aby przeszukiwanie się nie zapętlało. Przyjmij, że głębokość przeszukiwań jest ograniczona i zadawana każdorazowo przy inicjowaniu przeszukiwania. Również akcja, która ma być wykonywana dla każdej strony winna być parametrem wywołania odpowiedniej funkcji modułu.

Zaimplementuj rozwiązanie w formie iteratora, który zwraca wyniki działania funkcji będącej parametrem.

Zaprezentuj wykorzystanie modułu do wyszukiwania zdań zawierających słowo *Python*.

Zadanie 2. Napisz pakiet użytecznych narzędzi webowych, które pomogą pielęgnować i ulepszać prowadzony serwis:

1. moduł przeglądający strony WWW (pliki *.html) podanym katalogu i podkatalogach i sprawdzający, czy odnośniki do innych stron czy obrazków są aktywne;
2. moduł przeglądający strony serwisu (również przeglądając pliki) i dla każdej strony (pliku) wypisujący, w których plikach są odnośniki do niej.

Zaimplementuj te moduły jako iteratory zwracające wynik odpowiednich akcji.

Zadanie 3. Napisz własną miniwyszukiwarkę internetową, która

- przegląda strony i zapamiętuje liczbę wystąpień poszczególnych słów na poszczególnych stronach;
- zachowuje się podobnie jak pythonowy słownik, gdzie kluczem jest słowo, a wartością lista stron na których to słowo występuje (bądź lista pusta). Strony powinny być uszeregowane malejąco względem podanej liczby wystąpień. Możesz też zaproponować własną strategię rankowania stron.

Wystarczy, że indeksowane będą tylko wybrane strony, np. znajdujące się we wskazanych podkatalogach albo tylko strony do których da się dojść po odwołaniach w nie więcej niż k krokach od zadanej strony początkowej.

Na zajęcia należy wykonać jedno z tych zadań. Każde zadanie jest warte 4 punkty.

Marcin Młotkowski

6 listopada 2015

Kurs języka Python

Lista 6.

Zmodyfikuj zadanie z poprzedniej listy tak, aby poszczególne „podzadania” przeglądania stron były wykonywane w wątkach lub odrębnych procesach. W szczególności, aby operacje odczytu plików/pobierania stron znalazły się w odrębnych wątkach bądź procesach. Sprawdź, czy wykorzystywane w programie biblioteczne struktury danych są bezpieczne ze względu na wątki i odpowiednio zmodyfikuj operacje na nich, jeżeli nie nadają się do programów wielowątkowych.

Zadanie jest warte 4 pkt.

Marcin Młotkowski

20 listopada 2015

Kurs języka Python

Lista 7.

Zadanie 1. Zaprogramuj prosty program do rysowania za pomocą tzw. *grafiki żółwiowej*. Polega to na tym, że na ekranie (tj. w kontrolce ***DrawArea***) jest żółw, któremu możemy wydawać proste polecenia:

> IDŹ NAPRZÓD 20 pkt
> SKRĘĆ W MIEJSCU O 90 stopni
> IDŹ NA ŚRODEK

Żółw może chodzić po ekranie w dwóch trybach: zostawiając ślad i nie zostawiając śladu.

Polecenia żółwiowi powinne być wydawane za pomocą odpowiednich poleceń z menu, a parametry do tych poleceń zadawane poprzez kontrolkę `Entry`. Oczywiście nie jest konieczne rysowanie prawdziwego żółwia ;-)

Zadanie 2. Zaprogramuj prostą przeglądarkę obrazków (w formatach akceptowanych przez instytutową instalację Pythona). Przyjmij, że program może albo wyświetlać wszystkie obrazki z podanego (w ***Entry***) katalogu, albo poprzez jawne wskazanie pliku graficznego.

Zadanie 3. Zaprogramuj graficzny *minutnik*, tj. program odliczający wstecz zadany przez użytkownika czas i sygnalizujący koniec odliczania. Aby program był bardziej użyteczny, np. w kuchni, dodaj możliwość ustalania czasu odliczania poprzez wskazanie z menu pozycji typu *gotowanie ryżu* czy *gotowanie jajek na miękko*.

Zadanie 4. Zaprogramuj następującą prostą grę: na rysunku jest armata, która ma regulowany kąt wystrzału i prędkość początkową pocisku, oraz cel (odległość między armatą i celem może być losowa). Kąt wystrzału oraz prędkość pocisku powinna być zadawana przez użytkownika, np. za pomocą kontrolek ***Entry***. Zadanie polega na takim wybraniu kąta i prędkości, aby pocisk trafił w cel. Korzystając z prostych praw fizyki narysuj tor pocisku oraz oblicz, czy pocisk trafił w cel.

Zadanie 5. Bardzo ładnymi figurami geometrycznymi są fraktale. Sporo materiałów o nich można znaleźć w internecie (są również książki o fraktalach w naszej bibliotece). Zadanie polega na zaprogramowaniu kilku fraktali. Fraktale zwykle mają parametry, które powinne być podawane np. poprzez kontrolki ***Entry***.

Zadanie 6. Zaprogramuj program do rysowania dwuwymiarowych wykresów liniowych. Dane do wykresu są podawane w pliku zewnętrznym, na przykład w formacie CSV:

```
FS;FO;FOE;FSW;FZ;FPA;FRP;FANE;
1416.17;183.01;101.80;123.63;
1416.03;183.11;101.95;123.48;
1415.79;183.06;101.93;123.43;
```

Dane z pierwszego wiersza powinne być na osi X, a dane z kolejnych wierszy powinne być zaznaczane różnymi kolorami. Plik wejściowy powinien być wybierany z poziomu programu.

Zadania proszę wykonać za pomocą biblioteki Gtk i Cairo. Na zajęcia należy wykonać jedno z tych zadań. Każde zadanie jest warte 4 punkty.

Marcin Młotkowski

30 listopada 2015

Kurs języka Python

Lista 8.

Poniższe zadania polegają na implementacji aplikacji przechowujących trwale proste dane osobiste. Każda aplikacja powinna implementować:

- trwałe przechowywanie danych;
- graficzny interfejs użytkownika (np. w GTK+);
- wyświetlenie listy danych;
- elementarne wyszukiwanie danych spełniających jakiś prosty warunek;
- podstawowe operacje dodawania, odczytu, aktualizacji i usuwania pojedynczych danych[1].

Zadanie 1. Zaprogramuj własny organizator swojego czasu zawierający planowane spotkania (od-do), sprawy do załatwienia (do czasu), wraz z opcją przypominania.

Zadanie 2. Napisz program, który przechowuje w swojej lokalnej bazie danych informacje o posiadanych płytach z muzyką (identyfikator płyty, lista utworów i autorzy) wraz z informacjami o wypożyczeniu płyty znajomym.

Zadanie 3. Zaprogramuj własny notatnik z kontaktami do znajomych zawierający ich numery telefonów, adresy email czy datę ostatniego wyświetlenia tego kontaktu[2].

Na zajęcia należy wykonać jedno z tych zadań. Każde zadanie jest warte 6 punktów.

Marcin Młotkowski

[1] CRUD: create, read, update and delete

[2] Pozwoli to ustalić, z kim dawno się nie kontaktowaliśmy.

4 grudnia 2015

Kurs języka Python

Lista 9.

Wybierz jeden z wcześniejszych programów wykonanych w ramach pracowni z Pythona, ale nie starsze niż z listy 4. Wykonaj dla niego następujące zadania:

1. Przygotuj dla niego testy jednostkowe, można skorzystać z `pyunit` albo `pydoc`.

2. Sprawdź za pomocą profilowania, które fragmenty programu pochłaniają najwięcej czasu.

3. Poszukaj informacji o "PEP 8" (Python Enhancement Proposals). Za pomocą pakietu `pep8 checker`[3] sprawdź zgodność swojego kodu źródłowego z zaleceniami PEP 8.

4. Poszukaj informacji o automatycznym generowaniu dokumentacji na podstawie kodu źródłowego oraz zawartych w nim komentarzy. Wygeneruj taką dokumentację w jakims popularnym formacie (`html`, `pdf`, etc).

Zadanie jest warte 5 pkt.

Marcin Młotkowski

[3] zamiast instalować można skorzystać z serwisu online

11 grudnia 2015

Kurs języka Python

Lista 10.

Zadanie polega na modyfikacji programu z listy 8 tak, aby powstała aplikacja rozproszona. Obsługa zdarzeń interfejsu użytkownika zamiast wykonywać operacje na danych powinna wysyłać odpowiednie żądania do serwera, a zadania (np. modyfikacja bazy danych) powinien realizować serwer.

Zadanie jest warte 2 pkt. I jest ono ostatnie. Na stronie `http://www.ii.uni.wroc.pl/~marcinm/dyd/python/projekty.html` są opisane zasady dot. projektu końcowego wraz z przykładowymi tematami.

Marcin Młotkowski