Laboratorium kryptograficzne dla licealistów 1

Projekt „Matematyka dla ciekawych świata”
Łukasz Mazurek

23.03.2017

1 Praca z Pythonem

Na zajęciach będziemy programować w języku Python w wersji 3. Pythona można używać na jeden z dwóch sposobów:

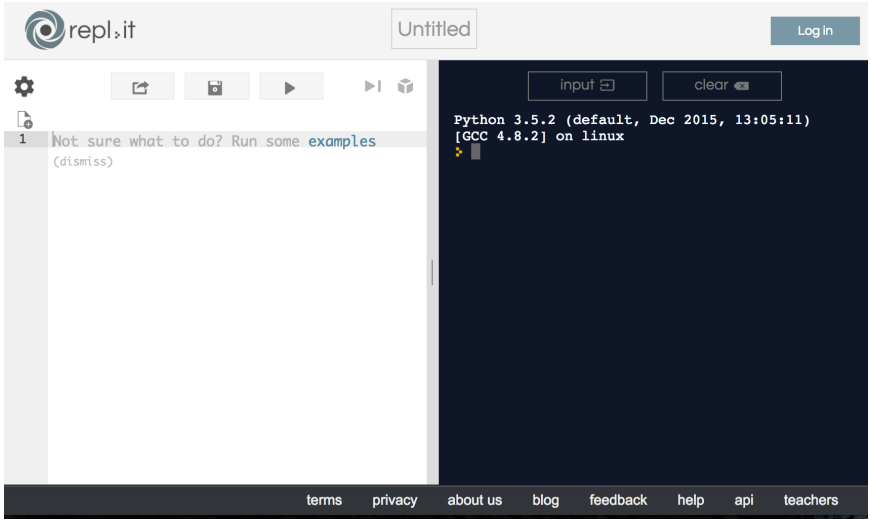
Zainstalowany interpreter Pythona. Interpreter Pythona jest zainstalowany na komputerach w laboratorium. Możecie go również bezpłatnie pobrać ze strony `https://www.python.org/downloads/` i zainstalować na swoich domowych komputerach. Zwróćcie uwagę, aby zainstalować wersję o numerze rozpoczynającym się od 3 (np. 3.6.0), a nie starszą, ale wciąż używaną wersję 2. Wersje te różnią się tak znacząco, że programy, które będziemy pisać na zajęciach nie będą działać w starszej, drugiej wersji Pythona.

Interpreter Pythona online. Istnieje wiele interpreterów Pythona online. Na zajęciach będziemy korzystali z interpretera znajdującego się na stronie `http://repl.it` . Interpretera tego będziecie mogli również używać na swoich domowych komputerach bez instalacji żadnego dodatkowego oprogramowania — wystarczy dowolna przeglądarka i podłączenie do internetu.

Innymi zasługującymi na uwagę interpreterami online są: `http://www.tutorialspoint.com/execute_python_online.php` i `http://ideone.com`

2 Praca z interaktywną konsolą Pythona `repl.it`

Uruchom przeglądarkę internetową i wejdź na stronę `http://repl.it`. W polu *Search for a language* na środku ekranu wybierz język *Python3*. Pojawi się okienko do zalogowania — można kontynuować bez logowania po kliknięciu *continue as Anonymous*. Powinieneś zobaczyć dwa okienka: miejsce na kod (białe, z lewej) i *konsolę* (czarne, z prawej, zwane również *terminalem*), tak jak na rysunku 1.



Rysunek 1: Widok interpretera Pythona dostępnego na stronie `http://repl.it`

Pierwszym sposobem pracy z Pythonem jest praca w interaktywnej konsoli, czyli praca w prawym okienku. W konsoli tej początkowo wypisane są pewne informacje (m. in. używana wersja Pythona) oraz znak zachęty `>`. Interpreter oczekuje, iż po tym znaku wpiszemy polecenie i naciśniemy Enter. Wynik polecenia zostanie wypisany w kolejnym wierszu (poprzedzony znakiem `=>`). Najprostszym sposobem użycia konsoli Pythona jest użycie jej jako kalkulatora — wpisujemy działanie do obliczenia, naciskamy Enter i w kolejnym wierszu otrzymujemy wynik działania. Przykład użycia konsoli Pythona jako kalkulatora znajduje się poniżej:

```
 > 2 + 2 * 2
=> 6
 > (2 + 2) * 2
=> 8
 > 2 ** 7
=> 128
 > 47 / 10
=> 4.7
 > 47 // 10
=> 4
 > 47 % 10
=> 7
```

W powyższym przykładzie:

- Znak `**` oznacza podnoszenie do potęgi.
- Znak `/` oznacza dzielenie.
- Znak `//` oznacza dzielenie całkowite.
- Znak `%` oznacza branie reszty z dzielenia.

Podobnie jak w kalkulatorze możemy korzystać z *pamięci*, w Pythonie możemy zapisywać wartości w *zmiennych*:

```
 > x = 3
 > y = 4
 > x
=> 3
 > x**2 + y**2
=> 25
```

W pierwszych dwóch linijkach następuje *przypisanie* wartości 3 do zmiennej `x` oraz wartości 4 do zmiennej `y`. Od tej pory możemy korzystać z tych zmiennych, np. do obliczenia wartości wyrażenia $(x^2 + y^2)$.

Zadanie 1 *Korzystając z Pythona jak z kalkulatora, znajdź wszystkie dziesięciocyfrowe potęgi dwójki.*

2.1 Pętla for

Załóżmy, że chcemy obliczyć kwadraty wszystkich liczb od 1 do 5. Zgodnie z dotychczasową wiedzą, w tym celu musimy wykonać 5 działań:

```
 > 1 * 1
=> 1
 > 2 * 2
=> 4
 > 3 * 3
=> 9
```

```
 > 4 * 4
=> 16
 > 5 * 5
=> 25
```

Widzimy jednak, że te działania są bardzo podobne i chciałoby się je wykonać „za jednym zamachem". Do wykonywania wielokrotnie tego samego (lub podobnego) kodu służą pętle. Najprostszym rodzajem pętli jest pętla **for**, która dla danej *listy* i operacji do wykonania wykonuje tę operację po kolei na każdym elemencie listy.

Do wykonania powyższego zadania służy pętla **for** w następującej postaci:

```
 > for x in [1, 2, 3, 4, 5]:
..     print(x * x)
..
1
4
9
16
25
```

Spróbuj przepisać tę pętlę do konsoli interpretera `repl.it` i uruchomić. Zwróć uwagę na kilka rzeczy:

- Na końcu pierwszej linijki jest dwukropek.
- Gdy naciśniemy Enter po zakończeniu pierwszej linijki, znak zachęty zmieni się z `>` na `..`, co oznacza to, że jesteśmy w trakcie pisania polecenia wielolinijkowego.
- Druga linijka musi być *wcięta*, tzn. rozpoczynać się od spacji, kilku spacji lub znaku tabulacji. Jeśli zapomnimy o wcięciu, interpreter zgłosi błąd i całą komendę wielolinijkową będziemy musieli pisać od początku (można pomóc sobie, naciskając strzałkę w górę i przeglądając stare komendy).
- Po wpisaniu drugiej linijki i naciśnięciu Entera pojawi się kolejny znak zachęty `..`, co oznacza, że interpreter czeka na ciąg dalszy polecenia wielolinijkowego. Jeśli nie chcemy pisać dalszego ciągu, w tym momencie musimy jeszcze raz nacisnąć Enter.

3 Pisanie i uruchamianie kodu programu

Do tej pory korzystaliśmy z Pythona używając interaktywnej konsoli (czyli prawego okienka). Jest to całkiem wygodne narzędzie, jeśli wykonujemy tylko jednolinijkowe polecenia, jednak pisanie dłuższych fragmentów kodu w tej konsoli staje się już bardzo niewygodne. Drugą metodą korzystania z Pythona jest pisanie kodu programu (skryptu) w pliku tekstowym (w lewym okienku) i uruchamianie tego kodu w konsoli (w prawym okienku).

Spróbuj teraz napisać w lewym okienku kod tej samej pętli for (pamiętając o dwukropku na końcu pierwszej i wcięciu drugiej linijki):

```
for x in [1, 2, 3, 4, 5]:
    print(x * x)
```

i wcisnąć przycisk „run" znajdujący się na górze tego okienka. Po chwili w konsoli po prawej powinien pojawić się wynik działania naszego programu:

```
1
4
9
16
25
```

Od tej pory właśnie w taki sposób będziemy pisać i uruchamiać wszystkie nasze programy. Ilekroć w niniejszych materiałach pojawią się dwie ramki, jedna obok drugiej, w lewej ramce znajdował się będzie kod programu, a w prawej efekt jego działania wyświetlony w konsoli:

```
for x in [1, 2, 3, 4, 5]:
    print(x * x)
```

```
1
4
9
16
25
```

Powyższa pętla wypisała każdą liczbę w osobnej linijce. Dzieje się tak, ponieważ polecenie **print**`(...)` domyślnie przechodzi do następnej linii po każdym wywołaniu. Można jednak zmienić to zachowanie, dodając wewnątrz **print**`(...)` po przecinku końcówkę `end = X`, gdzie `X` to otoczony apostrofami ciąg znaków, który chcemy wypisywać zamiast przejścia do nowej linii.

Przykłady:

```
for x in [1, 2, 3, 4, 5]:
    print(x * x, end = ' ')
```

```
1 4 9 16 25
```

```
for x in [1, 2, 3, 4, 5]:
    print(x * x, end = '')
```

```
1491625
```

```
for x in [1, 2, 3, 4, 5]:
    print(x * x, end = ' , ')
```

```
1 , 4 , 9 , 16 , 25 ,
```

3.1 Lista kolejnych liczb naturalnych

Często potrzebujemy, aby pętla przeszła po liście kilku kolejnych liczb naturalnych. W tym celu możemy oczywiście podać wprost kolejne elementy listy (tak jak w powyższym przykładzie), jednak istnieje wygodniejsze rozwiązanie, mianowicie polecenie **range**`()`:

```
for x in range(7):
    print(x, end = ', ')
```

```
0, 1, 2, 3, 4, 5, 6,
```

```
for x in range(5, 10):
    print(x, end = ', ')
```

```
5, 6, 7, 8, 9,
```

```
for x in range(10, 20, 3):
    print(x, end = ', ')
```

```
10, 13, 16, 19,
```

Na powyższych przykładach widzimy, że polecenie **range**`()` występuje w trzech wersjach:

- **range**`(kon)` generuje listę kolejnych liczb od 0 (**włącznie**) do `kon` (**wyłącznie**).
- **range**`(pocz, kon)` generuje listę kolejnych liczb od `pocz` (**włącznie**) do `kon` (**wyłącznie**).
- **range**`(pocz, kon, krok)` generuje listę liczb od `pocz` (**włącznie**) do `kon` (**wyłącznie**), przeskakując w każdym kroku o `krok`.

Do zapamiętania: ***Wszystkie przedziały w Pythonie są domknięte z lewej strony i otwarte z prawej strony, tzn. zawierają swój lewy koniec i nie zawierają swojego prawego końca.***

3.2 Słowa

Do tej pory używaliśmy zmiennych do przechowywania liczb i operowania na nich:

```
 > a = 2
 > b = 5
 > a + b
=> 7
 > a**b
=> 32
```

Zmienne mogą również jako wartości przyjmować litery, słowa, a nawet całe zdania:

```
 > x = 'A'
 > a = 'Ala'
 > b = "ma"
 > c = 'kota i wiele innych zwierząt'
 > x
=> 'A'
 > a[0]
=> 'A'
 > c[1]
=> 'o'
 > c[-1]
=> 't'
 > c[-3]
=> 'z'
 > a + b
=> 'Alama'
 > a + b + c
=> 'Alamakota i wiele innych zwierząt'
> 3 * a
=> 'AlaAlaAla'
```

Zwróć uwagę na następujące rzeczy:

- Napisy muszą być otoczone pojedynczymi apostrofami lub podwójnym cudzysłowami (nie ma znaczenia, którą wersję wybierzemy).
- Przy użyciu liczby w nawiasie kwadratowym możemy poznać poszczególne litery słowa (numeracja rozpoczyna się od 0).
- Ujemny indeks oznacza odliczanie liter od końca słowa: ostatnia litera słowa `c` to `c[-1]`, przedostatnia to `c[-2]`, itd.
- Przy użyciu znaku dodawania możemy sklejać (*konkatenować*) napisy.
- Przy użyciu znaku gwiazdki możemy mnożyć napisy (czyli sklejać same ze sobą).

Innymi przydatnymi operacjami na słowach jest sprawdzanie długości słowa poleceniem **len()** oraz wycinanie podsłowa przy użyciu dwukropka:

```
 > slowo = 'Python'
 > len(slowo)
=> 6
 > slowo[2:5]
=> 'tho'
 > slowo[3:]
=> 'hon'
 > slowo[:3]
=> 'Pyt'
```

W powyższym przykładzie:

- komenda `slowo[2:5]` zwraca podsłowo od znaku nr 2 (**włącznie**) do znaku nr 5 (**wyłącznie**),
- komenda `slowo[3:]` zwraca podsłowo od znaku nr 3 (**włącznie**) do końca,
- komenda `slowo[:3]` zwraca podsłowo od początku do znaku nr 3 (**wyłącznie**).

Tutaj też obowiązuje ta sama zasada, co w wypadku polecenia **range**`()`:

Wszystkie przedziały w Pythonie są domknięte z lewej strony i otwarte z prawej strony, tzn. zawierają swój lewy koniec i nie zawierają swojego prawego końca.

Korzystając z operacji wycinania podsłowa, możemy w łatwy sposób obliczyć ***rotację*** słowa, czyli słowo utworzone przez usunięcie kilku liter z początku słowa i przesunięcie ich na koniec. Na przykład druga rotacja słowa `'Python'` powstaje poprzez przesunięcie dwóch pierwszych liter tego słowa na koniec: `'thonPy'`. Efekt taki możemy uzyskać sklejając z sobą:

- podsłowo powstałe poprzez usunięcie dwóch pierwszych liter

oraz

- podsłowo złożone z tych dwóch pierwszych liter:

```
 > slowo = 'Python'
 > slowo[2:] + slowo[:2]
=> 'thonPy'
```

Zadanie 2 *Napisz pętlę, która wypisze wszystkie 6 rotacji słowa* `'Python'`*:*

```
Python
ythonP
thonPy
honPyt
onPyth
nPytho
```

3.3 Słowo jako lista

Wszystkie listy, których do tej pory używaliśmy w pętli **for** były listami liczb. Okazuje się, że w Pythonie pojedyncze słowo również jest listą, a dokładniej listą liter. Oznacza to, że po słowie można przejść przy użyciu pętli **for**, tak samo jak przechodziliśmy po liście liczb:

```
for l in 'Kula':
    print('litera', end = ' ')
    print(l)
```

```
litera K
litera u
litera l
litera a
```

Zwróć uwagę, że wewnątrz pętli for może znajdować się więcej niż jedno polecenie. Trzeba tylko pamiętać, aby wszystkie były poprzedzone takim samym wcięciem.

3.4 Lista słów

Poznaliśmy już dwa rodzaje list: listę liczb i listę liter (czyli słowo). Okazuje się, że w Pythonie możemy tworzyć listy obiektów dowolnego rodzaju, np. listy słów.

```
lista = ['Ala', 'ma', 'kota']
```

Podobnie jak w przypadku listy liczb i listy liter, po liście słów również możemy przejść pętlą for:

```
for slowo in lista:
    print(slowo)
```

```
Ala
ma
kota
```

W każdym kroku takiej pętli kolejne słowo z listy jest zapisywane do zmiennej `slowo`, a następnie zawartość tej zmiennej jest wypisywana na ekran. Oczywiście przed wypisaniem można dowolnie zmodyfikować zmienną `slowo`:

```
for slowo in lista:
    print(2 * (slowo + 'hh'), end = ' ')
```

```
AlahhAlahh mahhmahh kotahhkotahh
```

Na powyższym przykładzie widzimy, że:

- W kolejnych krokach pętli zmienna `slowo` zawiera kolejne słowa z listy.
- W każdym kroku pętli `slowo` jest sklejane ze słowem `'hh'`, a następnie mnożone przez 2 (czyli sklejane z samym sobą).
- Używając dodatkowego argumentu `end = ...` wymuszamy wypisanie spacji zamiast przejścia do nowej linii po końcu wypisywanego wyrażenia.

Zadanie 3 *Napisz pętlę, która dla danej listy słów* `lista` *utworzy akronim składający się z pierwszych liter każdego słowa. Np. dla listy* `['Matematyka', 'Informatyka', 'Mechanika']` *wypisze słowo* `MIM`.

4 Zadania dodatkowe

Zadanie 4 *Oblicz sumę* $1^2 + 2^2 + 3^2 + \ldots + 99^2 + 100^2$.

Zadanie 5 *Używając dwóch pętli* **for**, *jedna wewnątrz drugiej, napisz program, który wypisze na ekranie trójkąt z siódemek, taki jak poniżej:*

```
7
77
777
7777
77777
777777
7777777
77777777
777777777
```

Zadanie 6 *Napisz pętlę, która przejdzie po słowie zapisanym w zmiennej* `slowo` *wypisze je powtarzając każdą literę dwukrotnie. Np. dla* `slowo = 'kalafior'` *pętla powinna wypisać słowo* `'kkaallaaffiioorr'`.

5 Praca domowa nr 1

Rozwiązania zadań domowych należy przesłać do czwartku 30 marca do godz. 16^{59} na adres `licealisci.pracownia@icm.edu.pl` wpisując jako temat wiadomości `Lx PD1`, gdzie `x` to numer grupy, np. `L3 PD1` dla grupy L3, itd.

Zadanie domowe 1 (1 pkt) *Napisz pętlę, która wypisze wszystkie dwucyfrowe liczby podzielne przez 7. Kolejne liczby powinny być wypisane w jednym wierszu i porozdzielane pojedynczymi spacjami.*

Zadanie domowe 2 (2 pkt) *Napisz pętlę, która dla danej listy słów* `lista` *wypisze w kolejnych wierszach ich skróty w postaci* `<pierwsza litera>-<ostatnia litera> (<dlugosc slowa>)`. *Np. dla listy* `lista = ['Interdyscyplinarne', 'Centrum', 'Modelowania']` *powinna wypisać:*

```
I-e (18)
C-m (7)
M-a (11)
```

Wskazówka: wynik funkcji **len()** *mierzącej długość słowa jest liczbą. Do rozwiązania tego zadania może Ci się przydać konwersja tej liczby na słowo (aby dało się ją skleić z innymi słowami). Do tego służy polecenie* **str()**:

```
 > len('dudy')
=> 4
 > str(len('dudy'))
=> '4'
```

Zadanie domowe 3 (3 pkt) *Używając dwóch pętli* **for** *(jedna wewnątrz drugiej), napisz program, który dla danej listy słów* `lista` *wypisze każde słowo z listy wspak.*

Np. dla listy `lista = ['Ala', 'ma', 'kota']` *program powinien wypisać:*

```
alA
am
atok
```