ZAJĘCIA NR 1

Zawartość

Autor: ŁUKASZ CZERWIŃSKI

L.Czerwinski@students.mimuw.edu.pl
CzerwinskiLukasz1@gmail.com

2012-09-26

ZAJĘCIA 1

--- PODSTAWY PODSTAW (50 MINUT)

- jaki jest cel tworzenia programów komputerowych
- hasła, pierwsze logowanie,
- pierwsze uruchomienie konsoli Pythona, działania matematyczne
- wypisanie w konsoli stringa i liczby (komendą print)

Ważne terminy: konsola, linia poleceń, skrypt, bash, interpreter

➔ *Po co pisze się programy komputerowe*

Jaki jest cel pisania programów komputerowych?

Sensem pisania programów jest ułatwianie sobie życia. Człowiek jest leniwy, więc lubi, gdy ktoś za niego wykona jakąś pracę. Tak jest właśnie z programami komputerowymi - pisze się program (w domyśle: krótki), który po uruchomieniu wykona pracę, która ręcznie człowiekowi zajęłaby znacznie dłużej albo umożliwi człowiekowi korzystanie z pewnych udogodnień tam, gdzie przedtem ich nie było.

Przykłady:

- program komputerowy do obliczania toru pocisku i ustawiający odpowiednio działo w czołgu lub wyrzutnię w samolocie
  Kiedyś matematycy przez godzinę liczyli tor, uwzględniając masę pocisku, odległość od celu, ukształtowanie terenu, a także prędkość wiatru. Jeżeli przez tę godzinę wiatr się mocno zmienił, całe obliczenia szły do kosza! Teraz takie obliczenia zajmują komputerom sekundy.
- arkusz kalkulacyjny
  Podliczanie słupków 10 000 pozycji przy prowadzeniu firmy jest zajęciem czasochłonnym, a przy tym bardzo łatwo o pomyłkę. Arkusz Excel podlicza 10 000 liczb w mniej niż sekundę.
- projektowanie mostów
  W czasie projektowania mostów, budynków i innych konstrukcji inżynierowie muszą dokładnie policzyć, jakie siły działają na każdy element, żeby mieć pewność, że konstrukcja się nie zawali. Kiedyś liczyło się to tygodniami, sprawdzając po kilka razy każde obliczenie i mając nadzieję, że żaden błąd się nie prześlizgnął (stąd scena architekta stojącego pod mostem w trakcie próbnego przejazdu obciążonymi ciężarówkami była pełna napięcia i dramatyzmu). Obecnie programy komputerowe symulujące naprężenia elementów i siły fizyczne wykonują obliczenia w ciągu kilku dłuższych chwil.
- czytnik kodów kreskowych lub kodów QR w telefonie komórkowym
  Zalet chyba nie trzeba przedstawiać – ręczne wpisywanie długiego adresu lub wklepywanie kodu kreskowego jest mocno niewygodne.

➔ *Działania matematyczne*

- Działania:
  - dodawanie
  - odejmowanie
  - mnożenie
  - dzielenie (całkowite)
  - reszta z dzielenia
  - dzielenie niecałkowite (dla floatów)
  - potęgowanie

```
12+34
12-34
12*34
12/34  # dzielenie całkowite!
12.1/34
120%34
12**2
```

 * Zadanie 1.

Skoro 12/34 to dzielenie całkowite (z resztą), to jak zrobić dzielenie bez reszty?

 ** Zadanie 2.

Jak zrobić pierwiastkowanie?

 ** Zadanie 3.

Obliczyć NWD liczb 35 i 49, używając **algorytmu Euklidesa** i korzystając z działań w konsoli.

 *** Zadanie 3b.

Obliczyć NWD liczb 35 i 49, używając **szybkiego algorytmu Euklidesa** korzystając z działań w konsoli. (http://www.oeiizk.waw.pl/~witek/eliwww/strona2.html)

➔ *Pierwsza komenda: print*

```
print 1234
print 1234.56 # uwaga: kropka dziesiętna (przecinek rozdziela dwie
wartości)
print "ala ma kota"
print "ala", "ma", "kota" # przecinek przyda się później
```

--- UZUPEŁNIENIA PODSTAW

- Jak uruchomić gedit (kate lub inny)
- Skrypt w pliku – nagłówek i uruchamianie skryptu z Shella
- Komunikaty o błędach – jak wyglądają.

Ważne terminy: program, skrypt, algorytm.

➔ *Edytor tekstu*

W Linuksie jest wiele programów, które przypominają windowsowy Notatnik. Przykładami jest gedit i kate, ale jest też wiele innych, czasem z bardziej egzotycznym sposobem ich obsługi (vim, emacs).

➔ *Skrypt w pliku*

Należy pisać nagłówek:

```
#!/usr/bin/python
```

Najlepiej jeszcze od razu dodać obsługę polskich znaków:

```
# -*- coding: utf-8 -*-
```

Można także napisać bez „ozdobników”, np. tak:

```
# coding: utf-8
```

Jak uruchamiać:

```
python <nazwapliku>
```

➔ *Komunikaty o błędach*

Pamiętaj, że

- komunikaty są po angielsku
- podają nazwę pliku, numer linii i przyczynę (czasem błędnie, np. kolejną linię)

Kilka przykładowych błędów

- błędy składni:

```
pirnt "a"
print 3 "a"
```

- dzielenie przez zero (zadanie poniżej)
- polskie litery bez linijki `coding: utf-8` (zadanie poniżej)

Praktyczna rada

Warto parę razy przy różnych okazjach specjalnie popełnili jakiś błąd, żeby zobaczyć, jaki wyświetli się komunikat o błędzie. Dzięki temu później znacznie łatwiej szukać błędów, bazując na komunikatach o błędach – bo wszystkie błędy już się kiedyś widziało ☺.
Temu właśnie służą poniższe zadania.

 * Zadanie 1.

Sprawdź, jaki komunikat o błędzie Python wyświetli, gdy wpiszesz w programie jakieś nieistniejące słowo.

 * Zadanie 2.

Sprawdź, jak Python zareaguje na dzielenie przez zero.

 * Zadanie 3.

Sprawdź, co się stanie, gdy wkleisz do pliku .py program z polskimi literami, a nie napiszesz: `# -*- coding: utf-8 -*-`

 ** Zadanie 4.

Co się stanie, gdy wpiszesz nieprawidłowe znaki matematyczne, np.: 2++2, 2—2, 2+++2, 2---2, 2//2, 2///2, 2***2 itp. Sprawdź czy miałeś/miałaś rację.

 *** Zadanie 5.

Napisz dwa programy, które spowodują wyświetlenie jeszcze dwóch innych komunikatów o błędach. Przeczytaj komunikaty ze zrozumieniem.

--- ZMIENNE

http://docs.python.org/library/stdtypes.html

Co to jest zmienna?

Odpowiedź:
„Pojemnik na wartość”

Co może być wartością zmiennej?

Odpowiedź:
Liczba, napis, prawda/fałsz, a także typy złożone z innych, np. lista wartości (o których później) i in.

Zmiennej można przypisać wartość.

```
a = "Witaj!"
print a

a = 5
print a

a = 1.2
print a

a = True
print a
```

Do zmiennej można przypisywać nie tylko wpisaną przez programistę wartość, ale również: wynik działania, inną zmienną, działanie związanie z inną zmienną (innymi zmiennymi)

Na przykład:

```
a = 5
print a     # Wypisze: 5

b = a
print b     # Również wypisze: 5

c = a - 3
print c     # Wypisze: 2

c = 3 * b
print c     # Wypisze: 15
```

Co wypisze:

```
a = 5
a = a + 5
print a
```

Odpowiedź:
Wypisze: 10.

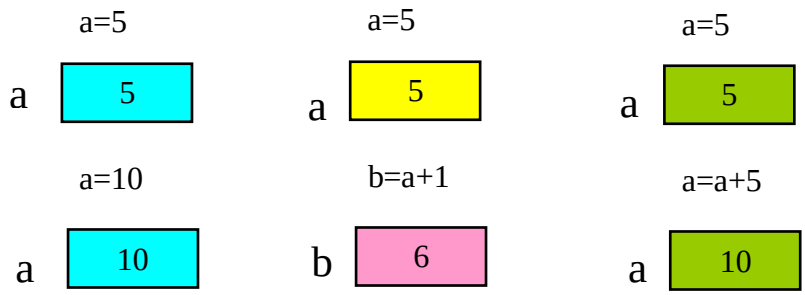
Uwaga!

Upewnij się, że rozumiesz, jak działa instrukcja:

```
a = a + 5
```

W razie wątpliwości przyjrzyj się dokładnie poniższemu schematowi.

Popatrz, jak działa przypisywanie do zmiennej:



* Zadanie 1.

Napisz program, który wypisze zdanie, przedstawiające Ciebie (np. „Mam na imię ...”), a następnie zmodyfikuj program tak, by każdy wyraz był w osobnej zmiennej.

* Zadanie 2.

Napisz program, który wypisze liczby: 2, 4, 6, 8 i 10, korzystając tylko z jednej zmiennej. Napisz go tak, żeby można było jednym ruchem zmienić program na wyświetlający: 3, 6, 9, 12 i 15.

--- TYPY I DZIAŁANIA

➔ *Podstawowe typy*

Każda wartość ma swój typ – a więc ma go też każda zmienna. Można go sprawdzić, pisząc:

```
type(2) # dla wartości
```

lub:

```
type(a) # dla zmiennych
```

Popatrzmy:

```
a = "Witaj!"
print a, type(a) # Witaj! <type 'str'>

a = 'Witaj!'
print a, type(a) # Witaj! <type 'str'>

a = 5
print a, type(a) # 5 <type 'int'>

a = 1.2
print a, type(a) # 1.2 <type 'float'>

a = True
print a, type(a) # True <type 'bool'>
```

Co pokaże:

```
a = 1.0
print type(a)
```

[int czy float?]

Odpowiedź:

Oczywiście: float (`<type 'float'>`)

Co pokaże:

```
type('')
```

Odpowiedź: `<type 'str'>`

Co pokaże:

```
type("")
```

Odpowiedź: `<type 'str'>`

Co pokaże:

```
type("3")
```

Odpowiedź: `<type 'str'>`

Typ None

Zarezerwowany z myślą o niezainicjowanych zmiennych

```
b = None
print type(b) # <type 'NoneType'>
```

➔ *Silne typowanie*

Przypomnijmy sobie dodawanie liczb:

```
print 3 + 4 # 7
```

Przypomnijmy sobie jeszcze łączenie napisów:

```
print "a" + "b" # ab
```

Jaki będzie efekt:

```
print 0 + "3" # nie zadziała - potrzebny str(), ale o tym za chwilę
```

Przeczytaj (ze zrozumieniem!) komunikat o błędzie.

str(), int() i float()

Żeby połączyć zmienne dwóch typów, w Pythonie trzeba je sprowadzić do tego samego typu (np. ‘str’, ‘int’ lub ‘float’). Dlatego w powyższym przykładzie trzeba skonwertować (zmienić typ) albo 0 (na string), albo "3" (na int lub float). Jak to zrobimy? Wykorzystując funkcje str(), int() i float() (o funkcjach będzie później, ale teraz nauczymy się korzystać z tych trzech). Robimy to w ten sposób:

```
print str(0)
print int("3")
print float("3")
```

Na oko nic się nie zmieniło – przynajmniej w sposobie wyświetlania, ale wyświetlmy typ str(0) i int("3") lub float("3"):

Jak wyświetlimy typ str(0) i int("3")?

```
print type(str(0))
print type(int("3"))
print type(float("3"))
```

Teraz widzimy, że str(0) ma typ ‘str’, int("3") ma typ ‘int’, a float("3") ma typ ‘float’.

* S Zadanie 1.

Wypisz wynik dodawania 0 i "3", korzystając z jednej z funkcji int, float lub str.
**Której lepiej użyć? Jaki wynik spodziewasz się otrzymać?

* S Zadanie 2.

Przerób powyższy przykład tak, aby zadziałał.

Rozwiązanie zad. 1 (szkic):

Nie ma odpowiedzi na pytanie, której funkcji lepiej użyć – wszystko zależy, co chcesz osiągnąć. Zapewne chcesz otrzymać 3, a więc użyj int() lub float(). Użycie str() da wynik 03 (i typ string!).

Co otrzymasz tutaj?

```
print "0" + "3" # oczywiście 03 (string!!)
```

Jak sprawdzić, że to string, a nie int?

Odpowiedź:

Użyć `type()`:

```
print type("0" + "3") # <type 'str'>
```

Co pokaże:

```
print "a" + 3
```

Odpowiedź:

Komunikat o błędzie.

Przeczytaj z uczniami komunikat o błędzie (ze zrozumieniem!).

Wniosek: To nie zadziała - potrzebny str().

Rozwiązanie zad. 2 (szkic):

Tutaj jest tylko jedno dobre rozwiązanie: str(), bo int() i float() nie zadziałają.

➔ *Priorytety działań*

Zastanów się, jaki będzie efekt poniższego działania. Sprawdź na komputerze. Dlaczego jest taki, a nie inny?

```
print 2 + 2 * 2
```

Odpowiedź:
Wynik to: 6. Jest to efektem działania **priorytetów operatorów**. Mnożenie jest wykonywane przed dodawaniem.

Wypróbuj wszystkie działania oraz nawiasy i na tej podstawie określ, jakie są priorytety działań.

Przykładowe działania:
2*3/2 (wyniki: 3 lub 2) lub porównanie: 2*(3/2) vs. 2*3/2
2+2-2 (wynik: 4 lub 2)
itp.

--- * LICZBY LOSOWE

Żeby korzystać z liczb losowych (a de facto: z liczb **pseudolosowych**), trzeba zaimportować dodatkowy moduł poleceniem:

```
import random
```

Następnie można korzystać z szeregu funkcji generujących liczby pseudolosowe:

```
random.random() # zwraca liczbę z przedziału [0, 1)
random.uniform(a, b) # zwraca liczbę z przedziału [a, b)
random.randrange(a, b) # zwraca liczbę całkowitą z przedziału [a, b)
```

Wyprzedzenie materiału

O listach będziemy mówić za jakiś czas, ale dla ambitnych wprowadzam je już tutaj.

```
random.choice(L) # zwraca losowy element z listy L
```

** Zadanie 1.

Zasymuluj rzut kostką do gry. Uruchom program kilka razy i sprawdź, że daje wszystkie możliwe wyniki.

** Zadanie 2.

Korzystając wyłącznie z funkcji random.random() wygeneruj liczbę z przedziału [0, 3).

** Zadanie 3.

Jak za pomocą funkcji random.random() zasymulować funkcję random.uniform(a, b)?

** Zadanie 4.

Jak za pomocą funkcji random.randrange(a, b) zasymulować random.choice(L)?

*** Zadanie 5.

Jak za pomocą funkcji random.random() zasymulować funkcję random.randrange(a, b)?

*** Zadanie 6.

Zasymuluj 6 rzutów kostką. Wypisz średnią arytmetyczną, minimum i maksimum wyników.