Mów mi Python!

Programowanie w języku Python

Po co, czyli cele

Celem zajęć jest zachęcanie nauczycieli i uczniów do programowania z wykorzystaniem języka Python. Przygotowane materiały prezentują zarówno zalety języka, jak i podstawowe pojęcia związane z tworzeniem programów i algorytmiką.

Oczekiwanym rezultatem szkoleń jest wyposażenie uczestników w minimum wiedzy i umiejętności umożliwiających samodzielne kodowanie w Pythonie.

Jak, a więc metody

Cechy języka Python przedstawiane są na przykładach, których realizacja może przyjąć różne formy w zależności od dostępnego czasu.

1. Prezentacja, czyli uruchamianie gotowych przykładów wraz z omówieniem najważniejszych fragmentów kodu.
2. Wspólne budowanie programów od podstaw: kodowanie w edytorze, wklejanie bardziej skomplikowanych fragmentów kodu.
3. Ćwiczenia w interpreterze Pythona – niezbędne m. in. podczas wyjaśnianiu elementów języka oraz konstrukcji wykorzystywanych w przykładach.
4. Ćwiczenia i zadania wykonywane samodzielnie przez uczestników.

Dla kogo, czyli co musi wiedzieć uczestnik

Dla każdego, co oznacza, że materiał zawiera moduły o różnym stopniu trudności. W zależności od poziomu wiedzy i zainteresowań uczestników przykłady można realizować wybiórczo.

Narzędzia

1. ***Interpreter Pythona*** w wersji **2.7.x**. Wykorzystanie wersji 3.x interpretera również jest możliwe, ale wymaga poprawiania kodu.
2. System operacyjny *Linux*, np. w wersji Live USB, dystrybucja dowolna, np. Ubuntu lub Debian. Python jest domyślnym składnikiem systemu.
3. System operacyjny *Windows XP / 7 / 8*. Interpreter należy doinstalować.
4. Dowolny edytor kodu, np. *Geany*, *PyCharm*, *Notepad++* (działają w obu systemach).
5. Narzędzia dodatkowe: *pip* (instalator modułów), *virtualenv* (zarządzanie środowiskami uruchomieniowymi), g*it* (zarządzanie rozproszonym systemem kontroli wersji). Trzeba doinstalować.
6. Biblioteki i frameworki Pythona wykorzystywane w przykładach: *Pygame, Peewee, SQLAlchemy, Flask, Django, Rgkit, RobotGame-bots*, *Rgsimulator*. Instalujemy w miarę potrzeb.

Co, czyli treści

1. **Podstawy Pythona**
   - Toto Lotek.

     Rozbudowany przykład wprowadzający podstawowe elementy języka, jak i programowania: zmienna, pobieranie i wyprowadzanie tekstu, proste typy danych, instrukcja warunkowa *if*, wyrażenie logiczne, pętla *for*, pętla *while*, *break*, *continue*, złożone typy danych, lista, zbiór, tupla, algorytm, poprawność algorytmu, obsługa wyjątków, funkcja, moduł.
   - Python w przykładach – zestaw przykładów prezentujących praktyczne wykorzystanie wprowadzonych zagadnień
2. **Gry w Pythonie** (Pygame)

   Przykłady multimedialne prezentujące tworzenie i manipulowanie prostymi obiektami graficznymi (Pong, Kółko i krzyżyk) oraz graficzną wizualizację struktur danych (Życie Conwaya).
   - Pong (wersja strukturalna i obiektowa)
   - Kółko i krzyżyk (wersja strukturalna i obiektowa)
   - Życie Conwaya (wersja strukturalna i obiektowa)
3. **Gra robotów** (Robot Game, rgkit)

   Przykład gry planszowej, w której zadaniem gracza-programisty jest tworzenie strategii walki robotów. Na podstawie przykładowych zasad działania robota oraz odpowiadającego im kodu, gracz "buduje" i testuje swojego robota. Zagadnienia: klasa, metoda, biblioteka, wyrażenia listowe, zbiory, listy, tuple, instrukcje warunkowe.
4. **Bazy danych** w Pythonie (SQLite)

   Przykłady wykorzystania bazy danych: model bazy, tabela, pole, rekord, klucz podstawowy, klucz obcy, relacje, połączenie z bazą, operacje CRUD (Create, Read, Update, Delete), podstawy języka SQL, kwerenda, system ORM, klasa, obiekt, właściwości.
   - Moduł SQL
   - Systemy ORM (Peewee i SQLAlchemy)
5. **Aplikacje internetowe**

   Przykłady zastosowania frameworków Flask i Django do tworzenia aplikacji działających w architekturze klient – serwer przy wykorzystaniu protokołu HTTP. Zagadnienia: żądania GET, POST, formularze, renderowanie widoków, szablony, tagi, treści dynamiczne i statyczne, arkusze stylów CSS
   - Quiz (Flask)
   - ToDo (Flask, SQLite)
   - Quiz ORM (Flask)
   - Czat (Django)

Przykładowe scenariusze

Zawartość treściowa scenariuszy oraz metody ich realizacji zależą przede wszystkim od dostępnego czasu. Zasada ogólna jest prosta: im więcej mamy czasu, tym więcej metod aktywizujących (kodowanie, testowanie, ćwiczenia, konsola Pythona, konsola Django itp.); im mniej, tym więcej

metod podających (pokaz, wyjaśnienia najważniejszych fragmentów kodu, kopiuj-wklej). W niektórych materiałach (np. gry w Pygame) po skopiowaniu i wklejeniu kodu warto stosować zasadę uruchom-zmodyfikuj-uruchom.

Poniżej przykładowy dobór i plan realizacji materiałów.

Warsztaty promocyjne (4,5 godz.)

1. **Podstawy Pythona** na przykładzie materiału *Toto Lotek*, punkty 5.1.1 do 5.1.10.

   Czas realizacji: 3 * 45 min.

   Metody: kodowanie programu w edytorze od podstaw, wprowadzanie elementów języka w konsoli interpretera, ćwiczenia samodzielne w zależności od poziomu grupy.

   Materiały i środki: scenariusz realizacji programu w wersji HTML, pełny kod programu, wersje pośrednie, jednak nie są konieczne, ponieważ kodu jest stosunkowo niedużo i każdy powinien nadążyć z kodowaniem. Niezbędny jest projektor. Dostęp do internetu nie jest konieczny.

   Realizacja:

   Na początku zapoznajemy użytkowników ze środowiskiem i narzędziami, tj. menedżer plików, edytor i jego konfiguracja, terminal znakowy, konsola Pythona, uruchamianie skryptu w terminalu, uruchamianie z edytora.

   Omawiamy założenia aplikacji "Mały lotek": losowanie pojedynczej liczby i próba jej odgadnięcia przez użytkownika. Następnie rozpoczynamy wspólne kodowanie wg materiału. Po skończeniu pierwszej części omawiamy założenia drugiej "Duży lotek": losowanie i zgadywanie wielu liczb. Realizujemy ją tylko do punktu 5.1.10 "Błędne dane!"

   Uwaga: obsługę błędnych danych można wprowadzić wcześniej.

   Po ukończeniu pierwszej i drugiej części po zakomentowaniu instrukcji drukujących losowane liczby można urządzić mini-konkurs, najlepiej z nagrodami :-)

   Budując program należy (!) reżyserować błędy implementacyjne i logiczne w jego działaniu. Uczestnicy powinni nauczyć się je dostrzegać, rozumieć i usuwać. Np. próba użycia liczby pobranej od użytkownika bez przekształcenia jej na typ całkowity, niewłaściwe wcięcia, brak inkrementacji zmiennej iteracyjnej (nieskończona pętla), itp.

   Uczymy dobrych praktyk programowania: przejrzystość kodu (odstępy), komentarze.

2. **Gra robotów** (Robot Game)

   Czas realizacji: 3 * 45 min.

   Metody: omówienie zasad gry, pokaz rozgrywki między przykładowymi robotami, kodowanie klasy robota z wykorzystaniem "klocków" (gotowego kodu), uruchamianie kolejnych walk.

Materiały i środki: przetłumaczona dokumentacja gry, strategia podstawowa oraz "klocki" w formacie HTML, końcowy kod przykładowego robota, koniecznie kody pośrednie. Niezbędny jest projektor. Wskazany dostęp do internetu lub wszystkie materiały offline w jednym archiwum dla każdego uczestnika.

Realizacja:

Na podstawie materiału Testowanie robotów omawiamy sposób przygotowania środowiska testowego, czyli użycie *virtualenv*, instalację biblioteki *rgkit*, *rgbots* i *rgsimulator*, polecenie *rgrun*.

Podstawą jest zrozumienie reguł. Najlepiej od razu uruchomić walkę między przykładowymi robotami z pakietu *rgbots* i prześledzić krok po kroku przynajmniej 11 rund. Uczestnicy powinni samodzielnie dostrzec i zrozumieć kilka zasad walki. Pozostałe reguły dopowiadamy i objaśniamy.

Zaczynamy od budowania strategii, czyli zaplanowania, co ma robot robić, wg jakich zasad ma walczyć. Dopiero później implementujemy. Wychodzimy od prostych reguł, stopniowo rozbudowując strategię.

Wyjaśniamy odwołania do api biblioteki *rg* w dodawanych "klockach". Kolejne wersje robota zapisujemy w osobnych plikach, tak aby można je było konfrontować ze sobą. Używamy symulatora robotów *rgsimulator*, aby wizualizować zaimplementowane strategie.

Zachęcamy uczestników do analizy zachowań robotów: co nam dało wprowadzenie danej zasady? jak moglibyśmy ulepszyć działanie robota?

Dodanie zbiorów pól jako podstawy logiki gry poprzedzamy przećwiczeniem w konsoli Pythona wyrażeń listowych, zbiorów i operacji na nich. Można wykorzystać przykłady z materiału "Python w przykładach → Nie znam Pythona... jeszcze". W kodzie można dodać polecenia drukowania poszczególnych zbiorów, co ułatwi ich zrozumienie.

Zazwyczaj nie ma potrzeby rozgrywania aż 100 rund, aby zobaczyć rezultaty dodawanych zasad. Korzystamy z *rg.settings.max_turns*, aby zmniejszyć ilość rund.

Podczas rozwijania kodu bardzo ważna jest kolejność podejmowania decyzji, sugerujemy uczestnikom eksperymenty.

Warsztaty szkoleniowe (ile godz.?)

[todo]