

Mów mi Python!

– czyli **programowanie w języku Python** w ramach projektu "Koduj z klasą" organizowanego przez Centrum Edukacji Obywatelskiej. Szczegóły pod adresem: <http://www.ceo.org.pl/pl/koduj>.

Po co, czyli cele

Celem projektu 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.

Ogólnym celem projektu jest propagowanie myślenia komputacyjnego, natomiast praktycznym rezultatem szkoleń ma być wyposażenie uczestników w minimum wiedzy i umiejętności umożliwiających samodzielne kodowanie w Pythonie.

Dla kogo, czyli co musi wiedzieć uczestnik

Dla każdego nauczyciela i ucznia, co oznacza, że materiał zawiera moduły o różnym stopniu trudności. Scenariusze zajęć oraz zakres przykładów można dostosować do poziomu uczestników.

Oprogramowanie

1. **Interpreter Pythona** w wersji **2.7.x**. Wykorzystanie wersji 3.x interpretera również jest możliwe, ale wymaga poprawiania kodu.
2. System operacyjny **Linux** w wersji **Live USB** – polecamy dedykowaną do szkoleń **LxPupTahr**, lub desktopowej, np. (X)Ubuntu lub Debian. Python jest domyślnym składnikiem systemu.
3. System operacyjny **Windows 7 / 8 / 10**. Interpreter Pythona należy doinstalować.
4. Dowolny edytor kodu, np. *Geany*, *PyCharm*, *Sublime Text* (działają w obu systemach).
5. Narzędzia dodatkowe: *pip* (instalator modułów), *virtualenv* (zarządzanie środowiskami uruchomieniowymi), *git* (zarządzanie rozproszonym systemem kontroli wersji).
6. Biblioteki i frameworki Pythona wykorzystywane w przykładach: *Matplotlib*, *Pygame*, *Peewee*, *SQLAlchemy*, *Flask*, *Django*, *Rgkit*, *RobotGame-bots*, *Rgsimulator*.

Uwaga: w ramach projektu przygotowano specjalną dystrybucję systemu Linux **LxPupTahr** przeznaczoną do instalacji na kluczach USB w trybie *live*. System zawiera wymagane narzędzia i biblioteki Pythona, umożliwia realizację wszystkich scenariuszy oraz zapis plików tworzonych przez uczestników szkoleń.

Co, czyli treści

Poniżej spis wszystkich materiałów przygotowanych w ramach projektu. Wersje źródłowe razem z kodami dostępne są w serwisie GitHub pod adresem: <https://github.com/koduj-z-klasa/python101>. Wersja HTML: <http://python101.readthedocs.org> lub <http://python101.rtf.d.org>.

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 kreśli (Matplotlib)*

Materiał prezentujący tworzenie wykresów oraz operacje matematyczne w Pythonie. Zagadnienia: listy, notacja wycinkowa, wyrażenia listowe, wizualizacja danych.

- *Python w przykładach* – zestaw przykładów prezentujących praktyczne wykorzystanie wprowadzonych zagadnień

2. 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.

3. 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)

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)

Jak, a więc metody

Cechy języka Python przedstawiane są na przykładach, których realizacja może przyjąć różne formy w zależności 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. Robot Game, gry w Pygame) po skopiowaniu i wklejeniu kodu warto stosować zasadę uruchom-zmodyfikuj-uruchom.

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.

Warsztaty promocyjne (4,5 godz.)

PODSTAWY PYTHONA

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.

Programy, materiały i środki: Python 2.7.x, edytor kodu, terminal, zalecany system Linux live *LxPupTahr*, wersja HTML scenariusza ***Toto Lotek, punkty 5.1.1 – 5.1.10***, kod pełnego programu oraz ewentualne wersje pośrednie. 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!" (przy czym 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 można reżyserować podstawowe błędy składniowe i logiczne, aby uczestnicy nauczyli się je dostrzegać 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) i komentarze.

WYKRESY W PYTHONIE

Czas realizacji: 1 * 30 min.

Metody: ćwiczenia w konsoli Pythona, wspólnie tworzenie i rozwijanie skryptów generujących wykresy, ćwiczenie samodzielne

Programy, materiały i środki: Python 2.7.x, biblioteka *Matplotlib*, edytor kodu, terminal, zalecany system Linux live *LxPupTahr*, wersja HTML scenariusza ***Python kreśli***. Projektor, dostęp do internetu nie jest konieczny.

Realizacja: Zaczynamy od prostego przykładu w konsoli Pythona, z której cały czas korzystamy. Stopniowo kodujemy przykłady wykorzystując je do praktycznego (!) wprowadzenia wyrażień *listowych* zastępujących pętle *for*. Pokazujemy również mechanizmy związane z indeksowaniem list, m. in. notację wycinkową (ang. *slice*). Nie ma potrzeby ani czasu na dokładne wyjaśnienia tych technik. Celem ich użycia jest zaprezentowanie jednej z zalet Pythona: zwięzłości. Jeżeli wystarczy czasu, zachęcamy do samodzielnego sporządzenia wykresu funkcji kwadratowej.

GRA ROBOTÓW (*Robot Game*)

Czas realizacji: 2 * 45 min. + 15 min.

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

Programy, materiały i środki: Python 2.7.x, biblioteka *rgkit*, przykładowe roboty z repozytorium *robotgame-bots* oraz skrypt *rgsimulator*, edytor kodu, terminal, zalecany system Linux live *LxPupTahr*, wersja HTML scenariusza ***Gra robotów***, końcowy kod przykładowego robota w wersji A i B, koniecznie (!) kody wersji pośrednich. Projektor, dostęp do internetu lub scenariusz offline w wersji HTML dla każdego uczestnika.

Realizacja: Na początku omawiamy [przygotowanie środowiska testowego](#), czyli użycie *virtualenv*, instalację biblioteki *rgkit*, *rgbots* i *rgsimulator*, polecenie *rgrun*. Uwaga: jeżeli korzystać będziemy z systemu *LxPupTahr*, w katalogu *~/robot* mamy kompletne wirtualne środowisko pracy.

Podstawą jest zrozumienie reguł. Po wyjaśnieniu najważniejszych zasad gry, konstruujemy robota podstawowego w oparciu o materiał [Klocki 1](#). Kolejne implementowane zasady działania robota sprawdzamy w symulatorze, ucząc jednocześnie jego wykorzystania. W symulatorze reżyserujemy również przykładowe układy, wyjaśniając szczegółowe zasady rozgrywki. Później uruchomiamy "prawdziwe" walki, w tym z robotami open source (np. *stupid26.py*).

Dalej rozwijamy strategię działania robota w oparciu o funkcje – [Klocki 2A](#) i/lub zbiory – [Klocki 2B](#). W zależności od poziomu grupy można przećwiczyć: tylko wersję A, A następnie B, A i B równolegle z porównywaniem kodu. Uwaga: nie mamy czasu na wyjaśnianie szczegółów implementacji, takich np. jak w wyrażenia zbiorów lub funkcje lambda.

Wprowadzając kolejne zasady, wyjaśniamy odwołania do api biblioteki *rg* w dodawanych "klockach". Kolejne wersje robota zapisujemy w osobnych plikach, aby można je było konfrontować ze sobą.

Zachęcamy uczestników do analizy kodu i zachowań robotów: co nam dało wprowadzenie danej zasady? jak można zmienić kolejność ich stosowania w kodzie? jak zachowują się roboty open source? jak można ulepszyć działanie robota?