Warsztaty 16 godz.

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.

Dla kogo, czyli co musi wiedzieć uczestnik

Dla nauczycieli pragnących wziąć udział w programie „Koduj z klasą” a w pierwszej edycji programu nie mieli takiej możliwości.

Cele, treści i metody

Cele projektu, spis wszystkich materiałów oraz zalecane metody ich realizacji dostępne są w dokumencie *Cele, materiały i metody*. Umieszczono tam również listę oprogramowania wymaganego do realizacji wszystkich materiałów. **Podstawą szkoleń jest *wersja HTML***. Wersje źródłowe dostępne są w repozytorium *Python101*.

Materiał zajęć

Środowisko programistyczne

Czas realizacji: 1 * 45 min.

Metody: uruchamianie menedżera plików, terminala, edytora, konfigurowanie edytora, uruchamianie interpretera i praca w konsoli Pythona.

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

Realizacja: Na początku zapoznajemy użytkowników z narzędziami. Uruchamiamy *menedżer plików* i wykonujemy kilka podstawowych operacji. Omawiamy działanie *terminala* (zwłaszcza dopełnianie i powtarzanie poleceń). Uruchamiamy i konfigurujemy (np. wcięcia 4 spacje) wybrany *edytor kodu*. Wspominamy o alternatywach. Uruchamiamy *konsolę Pythona* i wykonujemy kilka przykładów działań. Omawiamy *uruchamianie skryptów* w terminalu i z edytora (jeśli jest taka możliwość). Omawiamy *instalowanie Pythona i bibliotek* za pomocą narzędzi systemowych, jak i programu *pip*. Wyjaśniamy, w jaki sposób można przygotować bootowalny pendrive (odsyłamy do materiału Linux Live).

Toto lotek

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 1.2.1 – 1.2.14***, kod pełnego

programu oraz ewentualne wersje pośrednie. Projektor, dostęp do internetu nie jest konieczny.

Realizacja:

Omawiamy założenia każdej z części aplikacji, tj.: "Mały lotek" – losowanie pojedynczej liczby i próba jej odgadnięcia przez użytkownika; "Duży lotek" – rozwinięcie, losowanie i zgadywanie wielu liczb. Wspólnie kodujemy wg materiału. Nie stosujemy metody kopiuj-wklej (!). Uczestnicy samodzielnie wpisują kod, uruchamiają go i poprawiają błędy.

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.

Po ukończeniu pierwszej części można urządzić mini-konkurs: zgadnij wylosowaną liczbę.

Większość kodu (zgodnie z materiałem) ćwiczymy w konsoli, ucząc jej obsługi i wykorzystania.

Wykresy w Pythonie

Czas realizacji: 1 * 45 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 *wyrażeń 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 bądź innej.

Python w przykładach

Czas realizacji: 1 * 45 min.

Metody: ćwiczenia w konsoli Pythona, samodzielne wspólnie tworzenie i rozwijanie skryptów, ćwiczenia samodzielne.

Programy, materiały i środki: Python 2.7.x, edytor kodu, terminal, zalecany system Linux live *LxPupTahr*, wersja HTML scenariusza ***Python w przykładach*** i ***Pythonimzów***. Projektor, zalecany dostęp do internetu lub.

(pś, rb)

Realizacja: W zależności od zainteresowań grupy wybieramy jeden przykład spośród 1.4.5-1.4.9 do wspólnej realizacji, koncentrujemy się na utrwaleniu poznanych rzeczy, pokazaniu nowych. Jeśli się da, **wprowadzamy "pythonizmy"**, pokazując ich użycie w praktyce.

W przykładzie "Ciąg Fibonacciego" można pokazać rozwiązanie rekurencyjne. Przykłady "Słownik słówek" oraz "Szyfr Cezara" pozwalają wyeksponować operacje na tekstach i znakach, bardzo przydatne w rozwiązywaniu zadań typu maturalnego. "Oceny z przedmiotów" ilustrują operacje matematyczne, "Trójkąt" – przykładowe implementowanie algorytmu.

Gry w Pythonie

Czas realizacji: 2 * 45 min.

Metody: omówienie zasad gry, pokaz rozgrywki, kodowanie wykorzystaniem "klocków" (gotowego kodu), poprawianie błędów, optymalizacja.

Programy, materiały i środki: Python 2.7.x, biblioteka *pygame*, czcionka *freesansbold.ttf*, edytor kodu, terminal, zalecany system Linux live *LxPupTahr*, wersja HTML scenariusza ***Gry w Pythonie***, kody pośrednie i końcowy kod gry. Projektor, dostęp do internetu, jeżeli planujemy wykorzystanie serwisu GitHub do synchronizacji kodu lub scenariusz offline w wersji HTML dla każdego uczestnika.

Realizacja: Na początku omawiamy zasady gry w Ponga, pierwszej gry komputerowej (sic!). Kodowanie zaczynamy od wersji strukturalnej, wyjaśniając sposób tworzenia obiektów graficznych i manipulowania nimi. Posługujemy się metodą kopiuj-wklej. Zachęcamy uczestników do manipulowania właściwościami obiektów typu kolor, rozmiar itp.

Wyjaśniamy istotę działania programu z interfejsem graficznym opartego na pętli obsługującej zdarzenia (ang. *event driven apps*).

Następnie przechodzimy do wersji obiektowej, którą realizujemy krokowo metodą kopiuj-wklej wg scenariusza lub omawiamy kod końcowy. Wprowadzamy pojęcia klasa, obiekt (instancja), pole (atrybut) i metoda, konstruktor, pokazując naturalność traktowania graficznych elementów gry jako obiektów mających swoje właściwości (kolor, rozmiar, położenie) i zachowania (rysowanie, ruch), które można modyfikować. Odtwarzamy logikę i interakcje między obiektami: m. in. zastosowanie operatora * do przekazywania argumentów. Pokazujemy elegancję podejścia obiektowego, które wykorzystane zostanie w "Grze robotów" (sic!).

Jako ćwiczenie można zaproponować dodanie drugiej piłeczki i/lub zmianę orientacji pola gry: paletki po bokach.

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.

(pś, rb)

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. Przy wykorzystaniu klocków z zestawu B eksponujemy praktyczne zastosowanie *wyrażeń zbiorów i funkcji 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?

Bazy danych w Pythonie

Czas realizacji: 2*45 min.

Metody: równoległe kodowanie dwóch skryptów w edytorze, uruchamianie i testowanie wersji pośrednich, ćwiczenia z użyciem *interpretera Sqlite*.

Programy, materiały i środki: Python 2.7.x, biblioteka sqlite3 DB-API oraz framework Peewee, edytor kodu, terminal, zalecany system Linux live *LxPupTahr*, wersja HTML scenariusza ***SQL v. ORM*** *oraz* ***Interpreter Sqlite***, kody pełnych wersji obu skryptów. Projektor, dostęp do internetu lub scenariusz offline w wersji HTML dla każdego uczestnika.

Realizacja: Na początku pokazujemy przydatność poznawanych zagadnień: wszechobecność baz danych w projektowaniu aplikacji desktopowych i internetowych (tu odesłanie do materiałów prezentujących Flask i Django); obsługa bazy i podstawy języka SQL to treści nauczania informatyki w szkole ponadgimnazjalnej; zadania maturalne wymagają umiejętności projektowania i obsługi baz danych.

Na podstawie materiału równolegle budujemy oba skrypty metodą kopiuj-wklej. Wyjaśniamy

podstawy składni SQL-a, z drugiej eksponując założenia i korzystanie z systemów ORM. Pokazujemy, jak ORM-y skracają i usprawniają wykonywanie operacji CRUD oraz wpisują się w paradygmat projektowania obiektowego. Uwaga: ORM-y nie zastępują znajomości SQL-a, zwłaszcza w zastosowaniach profesjonalnych, mają również swoje wady, np. narzuty w wydajności.

Interpreter Sqlite wykorzystujemy do pokazania struktury utworzonych tabel (polecenia *.table*, *.schema*), później można (warto) przećwiczyć w nim polecenia CRUD w SQL-u.

Aplikacje internetowe

Czas realizacji: 4*45 min.

Metody: kodowanie wybranych aplikacji internetowych, uruchamianie i testowanie kolejnych, ćwiczenia samodzielne.

Programy, materiały i środki: Python 2.7.x, framework Flask i/lub Django, edytor kodu, terminal, zalecany system Linux live *LxPupTahr*, wersja HTML scenariusza ***Quiz*** i ***Czat***, kody wersji pośrednich i końcowych aplikacji. Projektor, dostęp do internetu lub scenariusz offline w wersji HTML dla każdego uczestnika.

Realizacja: Omówienie architektury klient-serwer jako podstawy działania aplikacji internetowych. Zaczynamy od scenariusza „Quiz”, który kodujemy metodą kopiuj-wklej. Wprowadzamy i wyjaśniamy pojęcia: protokół HTTP, żądanie GET i POST, kody odpowiedzi HTTP. Po uruchomieniu i przetestowaniu aplikacji pokazujemy jej prostotę, ale wskazujemy też ograniczenia: brak bazy danych, brak możliwości zarządzania użytkownikami, brak możliwości zmiany danych na serwerze.

Następnie realizujemy aplikację "Czat" wg scenariusza, stosując zasadę od znanego do nowego i nawiązując do wcześniejszych materiałów (SQL v. ORM i Quiz). Pokazujemy modułowość projektowania aplikacji, wynikającą z założeń wzorca MVC. Omawiamy projektowanie modelu bazy jako przykład zastosowania ORM w praktyce. Eksponujemy schemat dodawania stron: widok w views.*py* → szablon html → powiązanie z adresem w *urls.py*. Omawiamy dwa sposoby obsługi żądań: sprawdzanie w funkcji typu żądania i ręczne przygotowanie odpowiedzi oraz oparte na klasach widoki wbudowane automatyzujące większość czynności.