Mikrobitowcy

Autorzy: Filip Kłębczyk

Lekcja 1:

Pierwsze kroki z Pythonem. Python jako pomoc w prostych obliczeniach matematycznych

Podczas pierwszych zajęć uczniowie zapoznają się ze standardową implementacją języka Python. Najpierw dowiedzą się, jak realizować podstawowe operacje arytmetyczne i wykorzystywać zmienne z poziomu trybu interpretera, a następnie stworzą pierwsze programy w edytorze poznając sposoby interakcji z użytkownikiem - pobieranie danych, wypisywanie liczb i tekstów.

Cele lekcji:
Uczeń porafi:
- Znać pojęcia: zmienna, typ, wyjątek
- Wiedzieć, jak korzystać z trybu interaktywnego Pythona
- Wiedzieć, jak napisać i uruchomić prosty program wykonujący obliczenia matematyczne
- Wledzieć, jak wypisywać tekst
- Wledzieć, jak wczytać dane od użytkownika

Materiały pomocnicze:
- **https://codewith.mu/**
- **https://github.com/PyConPL/pyladies-workshop**
- **https://docs.python.org/**

Pojęcia kluczowe:
→ Python → zmienna → wyjątek → typ → operator

Czas realizacji: 45 min.

Metody pracy:
- wykład,
- dyskusja,
- ćwiczenia praktyczne przy komputerze,
- prezentowanie efektów pracy,
- szukanie rozwiązań na postawiony problem.

Treści programowe:
Podstawa programowa kształcenia ogólnego dla szkół podstawowych – II etap edukacyjny – klasy VII-VIII, informatyka:
I. Rozumienie, analizowanie i rozwiązywanie problemów. Uczeń:
3) przedstawia sposoby reprezentowania w komputerze wartości logicznych, liczb naturalnych (system binarny), znaków (kody ASCII) i tekstów; formułuje problem w postaci specyfikacji (czyli opisuje dane i wyniki) i wyróżnia kroki w algorytmicznym rozwiązywaniu problemów. Stosuje różne sposoby przedstawiania algorytmów, w tym w języku naturalnym, w postaci schematów blokowych, listy kroków;
5) prezentuje przykłady zastosowań informatyki w innych dziedzinach, w zakresie pojęć, obiektów oraz algorytmów.
II. Programowanie i rozwiązywanie problemów z wykorzystaniem komputera i innych urządzeń cyfrowych. Uczeń:
1) projektuje, tworzy i testuje programy w procesie rozwiązywania problemów. W programach stosuje: instrukcje wejścia/wyjścia, wyrażenia arytmetyczne i logiczne, instrukcje warunkowe, instrukcje iteracyjne, funkcje oraz zmienne i tablice.
III. Posługiwanie się komputerem, urządzeniami cyfrowymi i sieciami komputerowymi. Uczeń:
2) rozwija umiejętności korzystania z różnych urządzeń do tworzenia elektronicznych wersji tekstów, obrazów, dźwięków, filmów i animacji;
3) poprawnie posługuje się terminologią związaną z informatyką i technologią.
IV. Rozwijanie kompetencji społecznych. Uczeń:
1) bierze udział w różnych formach współpracy, jak: programowanie w parach lub w zespole, realizacja projektów, uczestnictwo w zorganizowanej grupie uczących się, projektuje, tworzy i prezentuje efekty wspólnej pracy;

Przygotowania przed zajęciami

Instalacja programowania: instalacja na wszystkich komputerach edytora **Mu** (https://codewith.mu/), alternatywnie skorzystanie z innego edytora.

Wprowadzenie w tematykę i integracja grupy (5 min.)

Na początku zajęć krótko przedstawiamy podstawowe informacje o języku Python. Jeśli uczniowie dotychczas programowali z wykorzystaniem wizualnych języków takich jak Scratch, objaśniamy również, że programując w Pythonie będą pisali kod w edytorze tekstowym, a nie tworzyli go wizualnie przeciągając i łącząc różne elementy.

Część zasadnicza (35 min.)

Wyjaśniamy uczniom różnice pomiędzy edytorem a trybem interaktywnym interpretera Pythona. Przedstawiamy zalety i wady obu rozwiązań - w edytorze możemy pisać wygodnie długie programy i łatwo wprowadzać poprawki, z kolei tryb interpretera nadaje się lepiej do pracy, gdy chcemy wykonać krótkie operacje lub przetestować działanie jakiejś funkcji lub fragmentu programu. Do obu podejść wystarczy nam zainstalowanie edytora Mu (https://codewith.mu/).

Pokazujemy sposób uruchomienia edytora - część do edycji kodu oraz opcję REPL, która uruchamia interaktywny interpreter. Na początku skorzystamy z interaktywnego interpretera, żeby zaprezentować podstawowe możliwości języka Python. Uświadamiamy uczniom, że mogą w nim sprawnie wykonywać operacje matematyczne, podobnie jak ma to miejsce w zwykłym kalkulatorze. Na początku prezentujemy trzy podstawowe operacje - dodawanie, odejmowanie, mnożenie, informując, że do dzielenia ze względu na specyfikę tej operacji przejdziemy później.

O ile w przypadku dodawania i odejmowania wykorzystujemy znaki **+** i **-** (jak ma to miejsce w zapisie matematycznym), to w przypadku mnożenia korzystamy z ***.

Pokazujemy parę typowych operacji (również takich, w wyniku których otrzymujemy liczby ujemne):

8 + 55

33 - 71

5 * 12

W interaktywnym interpreterze ich realizacja wygląda następująco:

```
In [1]: 8+55
Out[1]: 63

In [2]: 33-71
Out[2]: -38

In [3]: 5*12
Out[3]: 60
```

Prosimy uczniów o samodzielne podanie wyników poniższych operacji matematycznych, korzystając z interpretera Pythona:

a) 1343 × 6669 + 10500 × 123456789

b) 545 – 20 × 60 + 8

W oknie interpretera należy odpowiednio wpisać:

```
1343 * 6669 + 10500 * 123456789
```

i

```
545 - 20 * 60 + 8
```

Zwracamy uwagę na to, że Python w przeciwieństwie do zwykłego kalkulatora może operować na bardzo długich liczbach całkowitych - przykładowo nie jest problemem wykonanie działań składających się ze 100 cyfr (wiele kalkulatorów ma limit do kilkunastu cyfr lub mniej).

Podkreślamy też, że Python respektuje kolejność wykonywania działań, czyli dla powyższych przykładów najpierw wykonuje mnożenie, a potem dodawanie i odejmowanie (inaczej mówiąc o kolejności operacji decyduje priorytet operatorów). Jeśli chcemy zmienić kolejność operacji, musimy wykorzystać nawiasy okrągłe. Zwracamy uwagę, że niezależnie od zagnieżdżenia nawiasów stosujemy tylko nawiasy okrągłe (w języku Python nawiasy kwadratowe **[ ]** i klamrowe **{}** mają inne znaczenie i zastosowanie). Demonstrujemy to na przykładzie, pokazując jakbędzie wyglądał zapis w interpreterze poniższego wyrażenia:

{[5 × (3 + 2) – 8] + 7} × 3

W oknie interpretera należy wpisać odpowiednio:

```
((5 * (3 + 2) - 8) + 7) * 3
```

Prosimy uczniów o obliczenie następującego wyrażenia:

[30 **x** (10 – 42) + 84] × {[5 + 4 × (34 – 50)] + 5} × 3

W oknie interpretera uczniowie muszą wpisać w takim przypadku:

```
(30 * (10 - 42) + 84) * ((5 + 4 * (34 - 50)) + 5) * 3
```

Po tych ćwiczeniach przechodzimy do operacji dzielenia. Nauczyciel prezentuje proste dzielenie, tj. 5 ÷ 2

W oknie interpretera należy wpisać:

```
5 / 2
```

Otrzymany wynik 2.5 nie powinien zaskakiwać. Nauczyciel następnie prosi uczniów o wykonanie dzielenia 4 ÷ 2 i prosi uczniów o odpowiedź na pytanie, czy coś ich zaskoczyło lub zdziwiło w wyniku.

```
4 / 2
```

Nauczyciel tłumaczy, dlaczego otrzymano wynik nie 2 tylko 2.0. W przypadku dzielenia z wykorzystaniem operatora **/** w wyniku otrzymujemy zawsze liczbę zmiennoprzecinkową - typ float, czyli typ do reprezentowania w komputerze liczb rzeczywistych, nawet jeśli matematycznie wynik dzielenia jest liczbą całkowitą (czyli gdy liczba jest podzielna). Wcześniej operowaliśmy jedynie na liczbach całkowitych reprezentowanych przez typ int. Na tym etapie jedynie sygnalizujemy, że w Pythonie wartości 2 i 2.0 to wartości różnych typów, mimo, że reprezentują tę samą liczbę 2. Warto też nadmienić, że typ float reprezentuje liczbę rzeczywistą w przybliżeniu.

Następnie przedstawiamy uczniom operator dzielenia całkowitego **//** i operator modulo (reszty z dzielenia) **%**. Demonstrujemy przykładowe dzielenia z użyciem tych operatorów. Wpisujemy w interpreterze kolejno następujące wyrażenia:

```
4 // 2
```

```
4 % 2
```

```
5 // 2
```

```
5 % 2
```

Po tych przykładach pytamy uczniów, przez jaką liczbę w matematyce nie można dzielić drugiej liczby (odpowiedź: przez 0). Prosimy uczniów o wprowadzenie takiego dzielenia w interpreterze - korzystając zarówno z dzielenia operatorem **/** , jak i **//** . Obie operacje spowodują wystąpienie wyjątku **ZeroDivisionError.** Wyjątki to sposób sygnalizacji błędu, w tym konkretnym przypadku próby dzielenia przez 0.

Na koniec prezentujemy jeszcze operację potęgowania. Do tego celu wykorzystujemy operator ******.

5^2 zapiszemy w interpreterze jako:

```
5 ** 2
```

2^3 zapiszemy w interpreterze jako:

```
2 ** 3
```

Po tych przykładach prosimy uczniów o wykonanie samodzielnie następujących operacji (do dzielenia prosimy o stosowanie operatora **/**):

a) $10 \div 2 + 3$

b) $7^{\mathbf{2}} + 8^{\mathbf{2}}$

c) $(7 + 8)^{\mathbf{2}}$

d) $(43 + 28)^{\mathbf{2}} \times 50$

W interpreterze uczniowie powinni wpisać odpowiednio:

```
10 / 2 + 3
```

```
7 ** 2 + 8 ** 2
```

```
(7 + 8) ** 2
```

```
(43 + 28) ** 2 * 50
```

Następnie dajemy uczniom kolejne zadanie:

$$\frac{5 \times 31 + 2}{23 + 58} = ?$$

Prosimy uczniów o zastanowienie się, jakie operacje matematyczne tu zachodzą w jakiej kolejności. Następnie prosimy o wprowadzenie do interpretera kodu, który oblicza to wyrażenie. W rezultacie uczniowie powinni wprowadzić:

```
(5 * 31 + 2) / (23 + 58)
```

Potem nauczyciel stwierdza, że czasami chcemy zachować jakieś dane np. wyniki które są nam potrzebne do dalszych obliczeń. Zamiast zapisywać je na kartce lub gdzieś na komputerze możemy do tego celu użyć zmiennych. Objaśniamy, że zmienne to takie "nazwane pudełka przechowujące wartości, do których odwołujemy się po nazwie".

Na przykład chcemy wykonać kilka operacji z wykorzystaniem liczb 43540394 i 323054039, które oznaczymy sobie jako a i b.

a + 2 × b + b ÷ a

Następnie chcemy sprawdzić, ile to jest:

2 * a + b – 3 × (a + b)

W związku z tym, zamiast za każdym razem wpisywać lub wklejać liczby a i b, zapisujemy je do zmiennych, którym dla wygody nadajemy nazwy a i b . W języku Python jeśli chcemy zapisać wartość do zmiennej (inaczej mówiąc ustawić jej wartość), wykorzystujemy operator przypisania **=**. Potem możemy odwoływać się do wartości takich zmiennych wykorzystując ich nazwy. W interpreterze będzie to wyglądało następująco:

```
In [1]: a = 43540394

In [2]: b = 323054039

In [3]: a + 2 * b + b / a
Out[3]: 689648479.4196398

In [4]: 2 * a + b - 3 * ( a + b )
Out[4]: -689648472
```

Jak widać na powyższym przykładzie, najpierw przypisujemy wartości, a potem wykonujemy operacje. Po tej części prosimy uczniów o skopiowanie operacji z ostatniego przykładu (z wykorzystaniem myszy/touchpada) do edytora, zapełniając linie kodu w następujący sposób:

```
1  a = 43540394
2  b = 323054039
3  a + 2 * b + b / a
4  2 * a + b - 3 * ( a + b )
```

Informujemy uczniów, że teraz zrobimy to samo, ale nie w interpreterze, a w postaci kodu programu w edytorze. Chcemy też, by program wyświetlił na ekranie wyniki. W tym celu będziemy musieli skorzystać z funkcji **print** , gdyż w przeciwieństwie do interpretera w normalnych programach nie będziemy widzieć wyników wszystkich operacji. W nawiasie okrągłym (czyli jako parametr funkcji print) umieszczamy operację, której wynik chcemy wypisać. W efekcie program, który wypisze wyniki obu operacji, powinien wyglądać następująco:

```
1  a = 43540394
2  b = 323054039
3  print(a + 2 * b + b / a)
4  print(2 * a + b - 3 * (a + b))
```

Wybieramy opcję **Zapisz** i nadajemy nazwę programowi **prog1.py** (.py to rozszerzenie dla programów w języku Python), zamykamy **REPL** (klikamy REPL), żeby mieć więcej miejsca na oglądanie wyniku a następnie kilkamy przycisk **Wykonuj**. Teraz prosimy uczniów o podanie wyników dla operacji o innych a i b, np. 3 i 5, co uzyskuje się przez prostą modyfikację dwóch pierwszych linijek kodu:

```
1  a = 3
2  b = 5
3  print(a + 2 * b + b / a)
4  print(2 * a + b - 3 * (a + b))
```

Aby nasz program lepiej prezentował wyniki, możemy dodać tekst (ciąg znaków). W języku Python tekst umieszczamy pomiędzy parą cudzysłowów lub apostrofów:

```
"To jest tekst"
```

```
'I to jest tekst!'
```

Samo napisanie tekstu nam go nie wyświetli w wynikowym programie, dlatego tu też musimy skorzystać z funkcji **print** . Mając tę wiedzę możemy napisać:

```
1  a = 3
2  b = 5
3  print("Wynik pierwszego działania")
4  print(a + 2 * b + b / a)
5  print("Wynik drugiego działania!")
6  print(2 * a + b - 3 * (a + b))
```

Na tej zasadzie prosimy uczniów o napisanie własnego programu, który będzie obliczał obwód i pole powierzchni prostokąta o boku a równym 5 i b równym 8. Rozwiązanie może wyglądać następująco:

```
1  a = 5
2  b = 8
3  print("Obwód prostokąta:")
4  print(2 * a + 2 * b)
5  print("Pole prostokąta:")
6  print(a * b)
```

Bardzo często zamiast zapisywać dane w programie chcemy o nie zapytać użytkownika. W tym celu korzysta się z funkcji **input**. Przedstawiamy uczniom prosty program, który pyta się o imię, a potem wita nas żywając podanego wcześniej imienia.

```
1  print("Jak Ci na imię?")
2  name = input()
3  print("Cześć "+name)
```

Zwróćmy uwagę, że operator + służy tutaj do sklejania dwóch tekstów: "Cześć" oraz tekstu, który jest w zmiennej "name". Teraz możemy zmodyfikować nasz wcześniej napisany program do obliczania obwodu i powierzchni prostokąta. Dodatkowo przydatna będzie nam funkcja **int()**, która zamieni wpisaną liczbę w postaci tekstu (typ str) na typ liczby całkowitej (typ int). Przykładowy odczyt z konwersją ciągu znakowego na liczbę:

```
x = int(input())
```

Jeśli tego byśmy nie zrobili, przy dodawaniu tekst skleiłby się z innym tekstem. Natomiast mnożenie tekstu przez inny tekst spowodowałoby wystąpienie błędu - program zgłosiłby wyjątek:

```
TypeError: can't multiply sequence by non-int of type 'str'
```

Program z pobieraniem wartości a i b od użytkownika wygląda po zmianach następująco:

```
1  print("Podaj a:")
2  a = int(input())
3  print("Podaj b:")
4  b = int(input())
5  print("Obwód prostokąta:")
6  print(2 * a + 2 * b)
7  print("Pole prostokąta:")
8  print(a * b)
```

Ostatnie zadanie, które uczniowie mogą zacząć na lekcji i dokończyć w domu jest napisanie programu, który poprosi użytkownika o podanie liczby sekund i dla podanej wartości napisze, ile to razem minut i sekund (podpowiadamy uczniom, by wykorzystać operatory do dzielenia całkowitego i modulo).

Przykładowe rozwiązanie:

```
1  print("Podaj liczbę sekund:")
2  sec = int(input())
3  min = sec // 60
4  sec2 = sec % 60
5  print("Liczba minut to:")
6  print(min)
7  print("Liczba sekund to:")
8  print(sec2)
```

Możemy też zaprezentować wynik w przyjemniejszej dla oka formie minuty : sekundy. Zauważmy, że w ostatniej linii w celu rozdzielenia argumentów funkcji używamy przecinków.

```
1  print("Podaj liczbę sekund:")
2  sec = int(input())
3  min = sec // 60
4  sec2 = sec % 60
5  print(sec, "to", min, ":", sec2)
```

Podsumowanie (5 min.)

Nauczyciel krótko podsumowuje poznane elementy w trakcie lekcji. Zwraca uwagę uczniom, że nauczyli się tworzyć zmienne i wykorzystywać je w swoich programach oraz poznali trzy typy obiektów w Pythonie - int (typy dla liczb całkowitych), float (typ dla ułamków dziesiętnych) oraz str (typ łańcucha znakowego).

Uwagi

W języku Python zapisy...

```
a + b * 5
```

```
a+b*5
```

```
a + b*5
```

...są poprawne i robią dokładnie to samo. Spacje lub ich brak nie mają wpływu na samo działanie programu - jest to jedynie aspekt estetyczny tj. styl kodu. Na tym etapie najlepiej tylko delikatnie sugerować pewne dobre praktyki, skupiając się przede wszystkim na samym poprawnym działaniu programów, a potem ewentualnie na stylupisanego kodu. Zainteresowanych można odsyłać do dokumentu PEP-8 w dokumentacji języka Python, który zawiera zalecenia w tym względzie: **https://www.python.org/dev/peps/pep-0008/**

W trakcie lekcji warto też zwracać uwagę na przydatne elementy interpretera (klawisze kursora w górę/dół do przywoływania poprzednich operacji, uzupełnianie nazw zmiennych klawiszem TAB itd.).

Jeśli prowadzimy lekcję w sali, gdzie nie ma zainstalowanego edytora Mu lub interpretera, możemy skorzystać z przeglądarki i skorzystać z interpretera dostępnego na zdalnym serwerze w serwisie repl.it pod adresem: **https://repl.it/languages/python3**

Możemy też uczniom zasugerować korzystanie z tego rozwiązania, jeśli nie mają dostępu do komputera w domu, korzystają z komputera w bibliotece itp.