Mikrobitowcy

Autorzy: Filip Kłębczyk

Lekcja 4:

Pierwsze kroki z Pythonem. Pętle, wartości logiczne, listy

Podczas ostatniej lekcji z serii "Pierwsze kroki z Pythonem", uczniowie będą utrwalać dotychczasowe informacje oraz poznają nowe pojęcia: wartość logiczna, lista. Nauczą się także, jak dodawać komentarz w kodzie oraz jak zapisuje się wartości logiczne prawdy i fałszu. Będzie to ostatnia lekcja przed spotkaniem z płytką Micro:bit.

Cele lekcji:
Uczeń powinien:
- Znać pojęcia: wartość logiczna, lista, komentarz w kodzie
- Wiedzieć, co to jest pętla nieskończona
- Wiedzieć, jak zapisuje się wartości logiczne prawdy i fałszu w Pythonie
- Wiedzieć, jak działają operatory logiczne and, or oraz not
- Wykonywać podstawowe operacje na listach - wyświetlać,
- zliczać, dodawać, usuwać elementy
- Rozumieć do czego służą komentarze w kodzie i jak wyglądają

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

Pojęcia kluczowe:
→ wartość logiczna → lista

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;

Wprowadzenie w tematykę i integracja grupy (5 min.)

Pytamy uczniów, jakie poznali typy pętli na poprzedniej lekcji i czym się różnią. Pytamy również, z czym im się kojarzy pojęcie listy (np. lista zakupów, lista przystanków) i jakie właściwości są z nim związane (długość listy, kolejność elementów).

Część zasadnicza (40 min.)

Przedstawiamy uczniom następujący krótki kod programu.

```
1  while True:
2      print("Obieg")
```

O ile na poprzedniej lekcji po **while** wpisywaliśmy pewien warunek, który mógł być prawdziwy (co wiązało się z wykonaniem instrukcji wewnątrz pętli) lub fałszywy (co wiązało się z przerwaniem pętli), tutaj używamy wartości logicznej **True** (prawda). Jest to taki sam przypadek gdyby nasz warunek był cały czas spełniony (był prawdą), co oznacza, że program będzie wypisywał w kolejnych liniach słowo **Obieg** bez końca. Program wykonuje pętle nieskończoną, więc zazwyczaj nie ma szans się sam zakończyć (czasami operacje w pętli nieskończonej mogą spowodować błąd i tym samym przerwanie programu), dlatego możemy go przerwać wciskając CTRL--C, choć w edytorze Mu można też wybrać przycisk “Zatrzymaj”. W Pythonie wartości logiczne prawdy i fałszu to właśnie **True** i **False** . Są one bardzo często zwracane przez funkcje lub stosowane w wyniku porównań, a następnie przeważnie używane w pętlach lub instrukcjach warunkowych.

Uczniom przedstawiamy teraz trochę inny kod programu.

```
1  while True:
2      password = input("Podaj hasło: ")
3      if password == "stop":
4          print("Hasło poprawne")
5          break
6      else:
7          print("Błędne hasło")
8  print("Koniec programu")
```

Tym razem mamy pętlę, której warunek jest prawdą - zatem będzie wykonywała się ciągle, ale mamy możliwość jej przerwania. Z każdym obiegiem pętli użytkownik jest proszony o podanie hasła. Jeśli jest ono poprawne, wypisujemy informację o tym fakcie i przerywamy pętle z wykorzystaniem instrukcji **break**. W przeciwnym razie informujemy, że hasło jest błędne i pętla wykona się kolejny raz.

W języku Python możemy stosować trzy operatory logiczne **and**, **or** oraz **not**. Działają one tak, jak w innych językach programowania. **and** (operator logiczny “i”) przyjmuje wartość prawdy, tylko gdy oba argumenty są prawdą. **or** (operator logiczny “lub”) przyjmuje wartość prawdy, gdy przynajmniej jeden z argumentów jest prawdą. Operator **not** (logiczne “nie”) jest jednoargumentowy i neguje wartość logiczną (prawda staje się fałszem, a fałsz prawdą).

Nauczyciel przedstawia kolejny kod programu. Tym razem, żeby przerwać program, użytkownik musi wprowadzić dwa hasła. W tym celu zastosowano operator **and**, bo całe wyrażenie będzie prawdą, jeśli oba hasła **pass1** i **pass2** użytkownik wpisze poprawnie.

```
1  while True:
2      pass1 = input("Podaj pierwsze hasło: ")
3      pass2 = input("Podaj drugie hasło: ")
4      if pass1 == "stop" and pass2 == "koniec":
5          print("Hasła poprawne")
6          break
7      else:
8          print("Błąd - oba hasła muszą być poprawne!")
9  print("Koniec programu")
```

Prosimy uczniów o następujące modyfikacje powyższego programu:

a) Dodanie trzeciego hasła - użytkownik musi znać wszystkie trzy hasła by przerwać program

b) Wykorzystanie operatora logicznego **or** (logiczne "lub") tak, by wystarczyło, że użytkownik zna jedno z trzech haseł

Pierwsza modyfikacja mogłaby wyglądać następująco:

```
1  while True:
2      pass1 = input("Podaj pierwsze hasło: ")
3      pass2 = input("Podaj drugie hasło: ")
4      pass3 = input("Podaj trzecie hasło: ")
5      if pass1 == "stop" and pass2 == "koniec" and pass3 == "end":
6          print("Hasła poprawne")
7          break
8      else:
9          print("Wszystkie hasła muszą być poprawne!")
10 print("Koniec programu")
```

Druga modyfikacja to jedynie zamiana w linii 5 operatorów **and** na **or**...

```
5      if pass1 == "stop" or pass2 == "koniec" or pass3 == "end":
```

...oraz komunikatu w linii 9:

```
9          print("Wystarczy, że jedno hasło będzie poprawne!")
```

Następnie zadajemy uczniom kolejne zadanie - jest nim napisanie programu, który będzie losował liczbę z przedziału 1-100 a użytkownik będzie musiał odgadnąć, jaka to liczba (podejmując kolejne próby). Przy każdej próbie użytkownik będzie informowany przez program, czy wskazana liczba jest mniejsza czy większa od wylosowanej, czy może jest to dokładnie ta liczba. Jeśli trafienie jest poprawne, program wypisuje informacje o tym fakcie, a następnie podaje liczbę prób (za którym razem odgadnięto liczbę). Początek kodu programu do uzupełnienia udostępnia uczniom nauczyciel:

```
1  import random
2
3  number = random.randint(1, 100)  # liczba do odgadnięcia
4  guess = 0  # strzał użytkownika
5  counter = 0  # liczba prób
6
7  print("Zgadnij liczbę z przedziału 1-100")
```

Nauczyciel zwraca uwagę, że tekst po # to komentarze, czyli rodzaj notatek (inaczej mówiąc dokumentacji), jakie programiści zostawiają w kodzie programu. Nie są one przetwarzane przez komputer i nie mają wpływu na wykonanie programu - są tylko pomocą dla programisty lub zespołu programistów. Następnie podpowiada, że program będzie się wykonywał aż do momentu wskazania poprawnej liczby, o którą będzie pytał użytkownika w sposób ciągły. Przykładowe rozwiązanie:

```
import random

number = random.randint(1, 100)  # liczba do odgadnięcia
guess = 0  # strzał użytkownika
counter = 0  # liczba prób

print("Zgadnij liczbę z przedziału 1-100")
while True:
    guess = int(input("Twój strzał: "))
    counter += 1
    if guess < number:
        print("Za mała!")
    elif guess > number:
        print("Za duża!")
    else:
        print("Brawo, to jest ta liczba!")
        break
print("Liczba prób:", counter)
```

W drugiej części lekcji skupiamy się na listach i zachęcamy uczniów do eksperymentów na listach z wykorzystaniem trybu interaktywnego **REPL**. Lista (typ list) w Pythonie jest typem kontenerowym co oznacza, że możemy ją wykorzystywać do przechowywania elementów innych typów takich jak np. **int, float** czy **str** (elementem listy może być nawet inna lista). Na przykład listę składającą się z liczb całkowitych 1, 2, 3, 4, 5, którą przypiszemy do zmiennej **test_list** zapiszemy następująco:

```
test_list = [1, 2, 3, 4, 5]
```

Jak widać powyżej, początek i koniec listy oznaczamy nawiasami kwadratowymi, a poszczególne elementy oddzielamy przecinkiem. Równie dobrze możemy stworzyć listę składającą się z ciągów znakowych:

```
test_list2 = ["Ala", "Ola", "Jola", "Hela"]
```

Lub taką, która zawiera elementy różnych typów:

```
test_list3 = ['Ala', 5, 7.5]
```

By odwołać się do konkretnego elementu listy, musimy użyć indeksowania. W Pythonie - podobnie jak w wielu językach programowania - pierwszy element listy ma indeks 0, drugi indeks , trzeci indeks 2 itd. Musimy więc pamiętać, że jeżeli zależy nam n-tym elemencie listy, to stosujemy odpowiednio indeks n-1.

Przykładowo w REPL wyglądałoby to tak:

```
In [1]: test_list2 = ["Ala", "Ola", "Jola", "Hela"]

In [2]: test_list2[0]
Out[2]: 'Ala'

In [3]: test_list2[1]
Out[3]: 'Ola'

In [4]: test_list2[2]
Out[4]: 'Jola'

In [5]: test_list2[3]
Out[5]: 'Hela'
```

Możemy też odwoływać się do elementów od końca, stosując liczby ujemne. Ostatni element będzie miał wtedy indeks -1, przedostatni -2 itd. By odwołać się do ostatniego elementu listy test_list2 napiszemy:

```
test_list2[-1]
```

Jeśli chcielibyśmy naszą listę wzbogacić o jakiś element to możemy wykorzystać metodę (funkcję wywoływaną na rzecz obiektu) **append**. Doda ona element podany jako jej argument na koniec listy.

```
test_list2.append("Ela")
```

Lista **test_list2** będzie wtedy wyglądała następująco:

```
['Ala', 'Ola', 'Jola', 'Hela', 'Ela']
```

Aby usunąć element listy, korzystamy z metody pop . Jeśli nie podamy żadnego argumentu, to usunie ona i zwróci domyślnie ostatni element.

```
test_list2.pop()
```

Gdybyśmy chcieli ten element listy zachować w innej zmiennej, napisalibyśmy:

```
element = test_list2.pop()
```

Usunięcie pierwszego elementu wymaga z kolei podania indeksu 0 jako argumentu do metody **pop**.

```
test_list2.pop(0)
```

Musimy też pamiętać, że jeżeli lista będzie już pozbawiona elementów, to kolejne wywołanie **pop** zakończy się błędem. Co możemy jeszcze zrobić z listą? Możemy przykładowo sprawdzić jej długość, w tym celu korzystamy z funkcji **len**.

```
len(test_list)
```

Możemy też zliczyć liczbę elementów o konkretnej wartości występujących w liście z wykorzystaniem metody **count**. Przykładowa sesja z **REPL**, gdzie liczymy, ile razy występuje liczba 2 i 3 w liście, wygląda tak:

```
In [5]: test_list4 = [2,5,3,1,2,7,3,0,2]

In [6]: test_list4.count(2)
Out[6]: 3

In [7]: test_list4.count(3)
Out[7]: 2
```

Elementy listy możemy też łatwo wypisać stosując pętlę **for**, analogicznie jak było to w przypadku ciągów znakowych. Wyobraźmy sobie taki krótki program wypisujący wszystkie elementy listy (każdy w osobnej linii):

```
1  test_list4 = [2,5,3,1,2,7,3,0,2]
2
3  for elem in test_list4:
4      print(elem)
```

Możliwości i zastosowania list są naprawdę spore, więc powyższe przykłady zupełnie nie wyczerpują tego tematu. Warto zachęcać uczniów, by sami eksperymentowali, co można ciekawego robić z listami.

Wykorzystując poznane w czasie lekcji informacje, możemy napisać program przeprowadzający losowanie elementu z listy i usuwanie go, np. losowanie kolejności startu osób w zawodach sportowych. Poniżej zamieszczono przykładowy program. Pętla **while** (linia 4) wykonuje się dopóki długości listy jest różna od 0. Zastosowana jest tutaj też inna funkcja z modułu random - **choice**. Działa ona na takiej zasadzie, że losuje element z listy.

```
1  import random
2  name_list = ["Ola","Ala","Ela","Jola"]
3
4  while len(name_list) != 0:
5      name = random.choice(name_list)
6      print(name)
7      name_list.remove(name)
```

O ile zostanie nam czas, zadajemy uczniom następujące zadanie (które mogą rozwiązać też w domu): napisz program, który wylosuje 25 liczb z przedziału od 1 do 10, a na końcu poda, ile razy dana liczba została wylosowana. Aby rozwiązać to zadanie trzeba wylosować (najlepiej w pętli) 25 liczb, a następnie wypisać (najlepiej w kolejnej pętli) dla każdej z liczb od 1 do 10 liczbę wystąpień (stosując metodę **count** dla listy).

Uwagi

Lekcje 1-4 to zaledwie “dotknięcie” tematu programowania w języku Python. Ze względu na ograniczony czas ciężko zrealizować więcej materiału w trakcie tych paru lekcji. Następne rekomendowane kroki to zapoznanie się z:

- różnymi możliwościami wypisywania tekstu i różnymi operacjami na typie str
- odczytem i zapisem do pliku
- definiowaniem własnych funkcji
- obsługą wyjątków
- inne podstawowe typy kontenerowe - zbiór, słownik
- lista jako element listy i odwoływanie się do elementów listy wewnętrznej,
- zagnieżdżanie pętli.

Wiele z tych zagadnień można znaleźć w materiałach z warsztatów PyLadies organizowanych na konferencji PyCon PL i przeznaczonych dla początkujących: **https://github.com/PyConPL/pyladies-workshop**