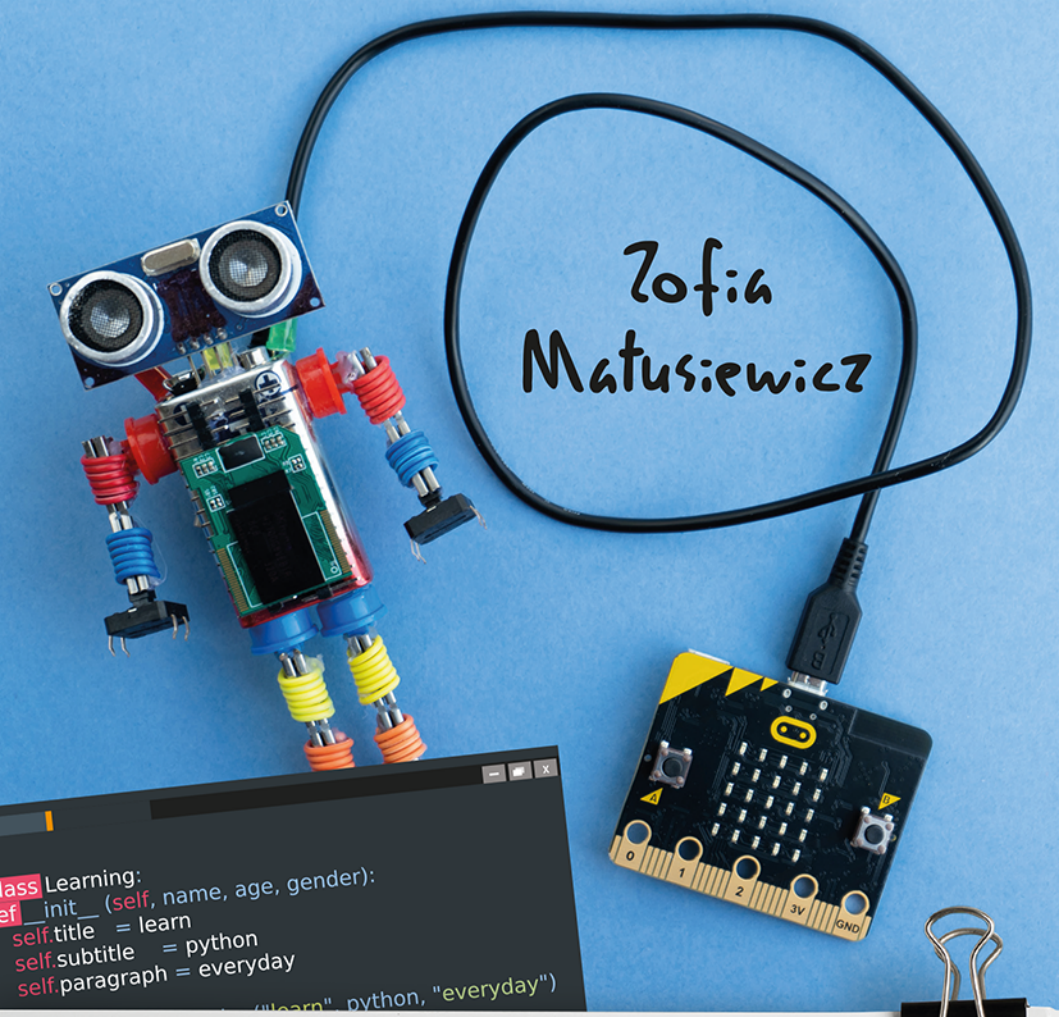


Zacznij od Pythona

PIERWSZE KROKI W PROGRAMOWANIU

Helion

Redaktor prowadzący: Szymon Szwajger

Grafika na okładce została wykorzystana za zgodą Shutterstock.com

Helion SA
ul. Kościuszki 1c, 44-100 GLIWICE
tel. 32 231 22 19, 32 230 98 63
e-mail: *helion@helion.pl*
WWW: *http://helion.pl* (księgarnia internetowa, katalog książek)

Drogi Czytelniku!
Jeżeli chcesz ocenić tę książkę, zajrzyj pod adres
http://helion.pl/user/opinie/zcpyth
Możesz tam wpisać swoje uwagi, spostrzeżenia, recenzję.

ISBN: 978-83-283-5820-1

Printed in Poland.

Spis treści

Rozdział 4.

Czym jest zmienna?

Umiesz już wyświetlać pewne informacje, ale ile razy można wypisywać te same teksty na ekranie? W sumie jeśli chcemy coś policzyć, to łatwiej włączyć kalkulator, a jeśli napisać, to przynajmniej notatnik. Czyż nie? Tak, masz rację. Dlatego program powinien być bardziej uniwersalny. Musi mieć jakiś ciekawy cel. Dojdziemy do tego krok po kroku. Kolejnym krokiem w tym kierunku jest wykorzystanie bytu, który nazywamy **zmienną**.

Być może ta nazwa kojarzy Ci się (słusznie) z matematyką, a z matematyką jest tak, że nie każdy ją lubi. Nie przejmuj się jednak. Mam nadzieję, że następująca analogia pomoże Ci zrozumieć pojęcie zmiennej. Rozwiązując zadanie z tekstem zaczynamy od wypisania szukanej, np. *x — ilość jabłek w koszyku*. Zmiennej *x* (bo to ona jest w tym przypadku zmienną) potrzebujemy do wykonania pewnych operacji, rozwiązywania równań, opisania wyniku.

Podobnie jest w programowaniu. Żebyśmy mogli wykonywać różne operacje, potrzebujemy zmiennych. W matematyce musimy na początku opisać, czym jest zmienna, zaś w programowaniu musimy mieć zarezerwowane takie miejsce w pamięci komputera, w którym będzie przechowywany aktualny stan wiedzy o zawartości zmiennej.

W programie może być zdefiniowanych bardzo wiele zmiennych — i to różnego typu. Oznacza to, że zmienne mogą przechowywać dane, które będą tekstem, liczbą całkowitą lub rzeczywistą (tzn. zmiennoprzecinkową), jedną z wartości logicznych — `prawda` lub `fałsz`.

W języku C++ w momencie tworzenia zmiennej trzeba podać jej typ, czyli wskazać, jakie dane będzie przechowywać. W Pythonie — nie. Zmienna może także zmieniać swoje typy, o czym powiemy za chwileczkę raz jeszcze. Po podaniu nazwy zmiennej przypisujemy jej wartość, a typ ustala się sam (dynamicznie).

Abyśmy mogli „dogadać się" z komputerem, o którą zmienną chodzi nam w danym momencie, musimy je w naszym programie ponazywać.

Prawidłowe nazwy zmiennych muszą spełniać pewne warunki:

- mogą składać się z następujących znaków: liter, podkreślnika (_) oraz cyfr,
- poszczególnych znaków nie oddzielamy spacjami,
- wielkość liter ma znaczenie,
- możemy używać cyfr pod warunkiem, że cyfra nie jest pierwszym znakiem w nazwie zmiennej.

Wybiegnijmy nieco do przodu i zobaczmy, jak wygląda działanie na zmiennych w programie. Przy okazji zwróć uwagę, że wyświetlając zmienną, nie używamy cudzysłowu.

Przykład 1. Nazwy zmiennych powinny sugerować, do czego będą służyć. Tutaj zmienna przechowuje liczbę, zatem jej nazwa to *liczba1* (bez spacji):

```
liczba1 = 3
print(liczba1)
```

Jak widzisz, do zmiennej przypisujemy konkretną wartość, podając najpierw nazwę zmiennej, następnie wpisując znak równości, a na końcu wartość, którą chcemy przypisać (`liczba1 = 3`). Kiedy to zrobimy, możemy odwoływać się do wartości zmiennej poprzez jej nazwę (`print(liczba1)`).

Oto wynik działania tego kodu:

```
3
```

Użycie spacji w nazwie zmiennej spowoduje błąd:

```
liczba 1 = 3
print(liczba 1)
```

Wynik działania to tym razem komunikat o błędzie:

```
File "<ipython-input-8-bfc16c774722>", line 1
liczba 1 = 3
```

```
^
SyntaxError: invalid syntax
```

Jeśli potrzebujemy, aby nazwa zmiennej składała się z wielu wyrazów, to każdy z nich oddzielamy podkreślnikiem, czyli znakiem „_”:

```
stolica_twojego_kraju = "Warszawa"
print(stolica_twojego_kraju)
```

Wynik działania:

```
Warszawa
```

Zamiast tego, możemy zaczynać poszczególne wyrazy (oprócz pierwszego) wielką literą (to tzw. *notacja camelCase*):

```
stolicaTwojegoKraju = "Warszawa"
print(stolicaTwojegoKraju)
```

Wynik działania jest oczywiście taki sam:

```
Warszawa
```

Zmienne, w których nazwach choć jedna litera jest różna, są uważane za różne (pamiętaj, że rozróżniamy także duże i małe litery):

```
kot = "dachowiec"
Kot = "Rudy"
print(kot)
print(Kot)
```

Wynik działania:

```
dachowiec
Rudy
```

Jak widzisz, `kot` i `Kot` to dwa różne koty!

Nazwa zmiennej może wprawdzie zwierać polskie znaki, jednakże lepiej ich nie stosuj. My też zazwyczaj nie będziemy tego robić, ten przykład jest pod tym względem wyjątkowy:

```
imię = "Janek"
print(imię)
```

Wynik działania:

```
Janek
```

Jak już widziałeś w przykładzie z kotem, zmiana wielkości liter w nazwie zmiennej jest przyczyną wielu błędów:

```
imie = "Janek"
print(Imie)
```

Wynik działania:

```
NameError Traceback (most recent call last)
<ipython-input-5-adf79d1ad46f> in <module>
1 imie = "Janek"
----> 2 print(Imie)
NameError: name 'Imie' is not defined
```

Błędy wykonania programu powoduje także zrobienie literówki w nazwie zmiennej. Poniżej utworzyliśmy zmienną `imię` (po raz kolejny zatem zastosowaliśmy polski znak — to naprawdę ostatni raz), a w instrukcji `print` próbujemy odwołać się do zmiennej `imie`, która w naszym programie nie istnieje:

```
imię = "Janek"
print(imie)
```

Wynik działania potwierdza wystąpienie błędu:

```
NameError Traceback (most recent call last)
<ipython-input-5-adf79d1ad46f> in <module>
   1 imię = "Janek"
----> 2 print(imie)
NameError: name 'imie' is not defined
```

W Pythonie ta sama zmienna może być redefiniowana, czyli raz być liczbą, a innym razem tekstem:

```
moja = 3
print(moja)
moja = "trzy"
print(moja)
```

Wynik działania:

```
3
trzy
```

Użycie niedozwolonego znaku w nazwie zmiennej spowoduje błąd:

```
jakas!zmienna = 3
```

Wynik działania:

```
File "<ipython-input-8-bfc16c774722>", line 1
jakas!zmienna = 3
^
SyntaxError: invalid syntax
```

Typ zmiennej można sprawdzić używając funkcji `type`, podając jej nazwę zmiennej — zgodnie ze składnią `type(nazwaZmiennej)`, przy czym zamiast `nazwaZmiennej` wpisujemy, oczywiście, nazwę zmiennej, której typ chcemy sprawdzić.

Dla tekstu, liczb i wartości logicznych mamy odpowiednie, następujące typy: **`str`** (tekst), **`int`** i **`float`** (liczby) oraz **`bool`** (wartości logiczne).

```
tekst = "Ja"
calkowita = 7
rzeczywista = 2.3
logiczna = True
print(type(tekst))
print(type(calkowita))
print(type(rzeczywista))
print(type(logiczna))
```

Wynik działania:

```
<class 'str'>
<class 'int'>
<class 'float'>
<class 'bool'>
```

Łańcuchy

Upraszczając nieco, można powiedzieć, że łańcuch (ang. *string*) to tekst. Znaki mają swoją kolejność, a cały łańcuch długość. W C++ jest oddzielny typ na jeden znak, a w Pythonie — nie. Jeden znak to tylko łańcuch o długości jednego znaku.

Łańcuchy możemy przede wszystkim wprowadzać i wyświetlać (o czym będzie powiedziane bardziej szczegółowo w następnym rozdziale).

Wyświetlmy przykładowy tekst, używając zmiennej:

```
mojTekst = "Jak się czujesz?"
print(mojTekst)
```

Oto wynik działania tego kodu:

```
Jak się czujesz?
```

Możemy też to zrobić, jak widziałeś to już wcześniej, bez używania zmiennej:

```
print ("Jak się czujesz?")
```

Wynik działania jest taki sam:

```
Jak się czujesz?
```

Łańcuchy możemy łączyć. Używamy do tego operatora „+”:

```
powitanie = "Cześć, jak się masz "
imie = "Piotrek"
powitanieSpersonalizowane = powitanie + imie
print(powitanieSpersonalizowane)
```

Wynik działania:

```
Cześć, jak się masz Piotrek
```

Zauważ, że na końcu zmiennej `powitanie`, przed zamknięciem cudzysłowu, wpisaliśmy spację. Gdybyśmy tego nie zrobili, w wyniku połączenia łańcuchów `powitanie` i `imie` otrzymalibyśmy ciąg wynikowy w postaci:

```
Cześć, jak się maszPiotrek
```

Możemy wyświetlać pojedyncze znaki łańcuchów — pisząc w nawiasie kwadratowym, który (licząc od 0 — tak zazwyczaj liczy się w programowaniu pozycje, m.in. w tablicach, o których będzie jeszcze mowa) znak chcemy wyświetlić.

Zdefiniujmy prostą zmienną łańcuchową `mebel`, przypiszmy jej wartość `stół`, po czym wyświetlmy znaki znajdujące się na pozycji zero i jeden:

```
mebel = "stół"
print(mebel[0])
print(mebel[1])
```

Wynik działania:

```
s
t
```

Pamiętajmy, że nie możemy zmieniać łańcucha poprzez podmianę znaków (co jest możliwe np. w C++):

```
mebel = "stół"
mebel[2] = "x"
```

Oto wynik tego kodu:

```
TypeError Traceback (most recent call last)
<ipython-input-20-1c12e950cfe1> in <module>
1 mebel = "stół"
----> 2 mebel[2] = "x"
TypeError: 'str' object does not support item assignment
```

Liczby całkowite

Przypomnijmy, że liczby całkowite to liczby: 0, 1, 2, 3 itd., oraz liczby do nich przeciwne (czyli ujemne — z minusem). W Pythonie typem dla liczb całkowitych jest `int` (od ang. *integer* — całkowita).

O operacjach, jakie wykonujemy na liczbach, powiemy bardziej szczegółowo w dalszej części książki. Jednak już teraz spróbujmy przeanalizować prosty przykład.

Policzmy różnicę dwóch liczb całkowitych. Wynik jest również liczbą całkowitą:

```
liczba1 = 5
liczba2 = 3
roznica = liczba2 – liczba1
print(roznica)
print(type(roznica))
```

Wynik działania:

```
2
<class 'int'>
```

Ale już wynik dzielenia tych samych liczb jest innego typu, określającego liczby rzeczywiste, czyli `float`, o którym będziemy mówić za chwilę.

```
liczba1 = 5
liczba2 = 3
iloraz = liczba2 / liczba1
print(iloraz)
print(type(iloraz))
```

Wynik działania:

```
0.6
<class 'float'>
```

Jeśli próbowałeś programować w C++, zapewne widzisz tu koleją różnicę między tym językiem a Pythonem. W C++ w wyniku podzielenia dwóch liczb całkowitych mamy liczbę całkowitą, zaś w Pythonie jest to liczba rzeczywista (co jest zgodne z naszą intuicją).

Musisz oczywiście pamiętać, że 3 jest liczbą całkowitą, zaś `'trzy'` oraz `"trzy"` łańcuchami. Próba dodania łańcucha do liczby (i na odwrót) spowoduje błąd.

Nie dodajemy liczb do łańcucha. Spójrz na przykład pokazujący, co się stanie, jeśli spróbujemy to zrobić:

```
a = "trzy"
b = 3
c = a + b
```

Wynik działania:

```
TypeError Traceback (most recent call last)
<ipython-input-25-def6741182aa> in <module>
1 a = "trzy"
2 b = 3
----> 3 c = a + b
TypeError: can only concatenate str (not "int") to str
```

Musisz też uważać, bo łańcuchy wcale nie muszą zawierać liter — `"3"` (zapisane w cudzysłowie) to także łańcuch, a nie liczba. W związku z tym wykonanie takiego działania:

```
a = "3"
b = 3
c = a + b
```

zwróci taki sam błąd jak poprzednio:

```
TypeError Traceback (most recent call last)
<ipython-input-24-c696c82f222f> in <module>
1 a = "3"
2 b = 3
----> 3 c = a + b
TypeError: can only concatenate str (not "int") to str
```

Liczby zmiennoprzecinkowe

Liczby zmiennoprzecinkowe są to liczby rzeczywiste, czyli takie, które posiadają część ułamkową (która — uwaga — może być również równa zeru, co nie zmienia typu tej liczby czy zmiennej ją reprezentującej). Mogą to być np. liczby: 3,6, 5,7234, 11,0. Przy pisaniu programów wykorzystujących liczby rzeczywiste musisz pamiętać jeszcze o jednym. Podczas gdy w języku polskim używamy przecinka do oddzielenia liczby od części ułamkowej (3,5), w angielskim używa się do tego kropki (3.6). I to właśnie taki zapis — z kropką — musisz stosować w edytorze Pythona oraz wszędzie, gdzie piszesz kod w tym języku.

Typ zmiennej dla liczb rzeczywistych to `float`. Dodajmy jeszcze, że liczby zmiennoprzecinkowe, czyli rzeczywiste, są prezentowane w pamięci komputera z pewną dokładnością.

Przykład 1. Wiemy, że $\frac{1}{3}$ (czyli 1/3) jest w gruncie rzeczy liczbą nieskończoną, ale w programie otrzymamy konkretną, zależną od ustalonej precyzji, liczbę cyfr po przecinku.

```
liczba = 1/3
print(liczba)
```

Wynik działania przy domyślnej precyzji to:

```
0.33333333333333333
```

Przykład 2. Liczba, w której nie podaliśmy części ułamkowej, jest traktowana jako liczba całkowita, zaś ta, która ją posiada, jest liczbą zmiennoprzecinkową.

```
a = 5
print(type(a))
b = 5.0
print(type(a))
```

Wynik działania:

```
<class 'int'>
<class 'float'>
```

Zwróć uwagę, że w liczbie 5.0 zawarta jest część ułamkowa, choć w tym przypadku jest ona równa 0.

Liczba typu `float` może być wynikiem działań na liczbach całkowitych — dzieje się tak w przypadku dzielenia.

Wartości logiczne

Typ logiczny to typ danych, który ma tylko dwie wartości: prawdę (`True`) oraz fałsz (`False`). Jest on jednak bardzo ważny, dzięki niemu wykonuje się cały szereg czynności w programowaniu.

Przykład 1. Typem zmiennej logicznej jest `bool`:

```
prawda = True
print(type(prawda))
```

Wynik działania:

```
<class 'bool'>
```

Typ logiczny używany jest między innymi po to, aby stwierdzić prawdziwość jakiegoś warunku.

Przykład 2. Zobaczmy, co się wyświetli w wyniku sprawdzenia, czy liczba 3 jest większa od 5:

```
print(3 > 5)
```

Wynik działania:

```
False
```

Za pomocą operatorów logicznych, które poznasz bardziej szczegółowo w dalszych rozdziałach, możemy budować także złożone wyrażenia.

Przykład 3. Sprawdźmy, czy 5 jest równe 6 lub 6 jest mniejsze niż 9:

```
print(5 > 6 or 6 < 9)
```

Wynik działania:

```
True
```

Nie przejmuj się, jeśli nie wiesz, co znaczy użyty w tym przykładzie operator `or`. Podobnie jak `and` w następnym. Zostanie to później wyjaśnione.

Przykład 4. Zobaczmy, czy 6 jest większe niż 7 i 4 jest mniejsze niż 7:

```
print(6 > 7 and 4 < 7)
```

Wynik działania:

```
True
```

Przykład 5. Porównajmy teksty, używając operatora porównania (`==`):

```
a = "Ala"
b = "ala"
c = "Ala"
print(a == b)
print(a == c)
```

Wynik działania:

```
False
True
```

Jest niezwykle ważne, żebyś pamiętał, że do porównywania służy operator zapisywany jako dwa kolejne znaki równości (==), a nie jeden taki znak (=); pojedynczy znak równości, jak być może pamiętasz, używany jest do przypisywania do zmiennej określonej wartości.

Na koniec podrozdziału warto wspomnieć o jeszcze jednym fakcie, który pozwoli nam sprawniej operować na zmiennych: możemy im przypisywać nie tylko wartości liczbowe czy tekstowe, ale również wartości innych zmiennych.

Niech `a = 2`, zaś `b = 9`. Chcemy zamienić wartości zmiennych, czyli sprawić, by w zmiennej `a` było to, co jest w `b`, zaś w `b` to, co jest w `a`:

```
a = 2
b = 9
# w zmiennej o nazwie pomocnicza przechowamy to, co teraz jest w zmiennej a
pomocnicza = a
# do zmiennej a przypiszemy to, co jest w zmiennej b
a = b
# do b przypiszemy teraz to, co jest w zmiennej pomocniczej (czyli pierwotną wartość zmiennej a)
b = pomocnicza
print(a)
print(b)
```

Wynik działania:

```
9
2
```

Znak # oznacza, że w tym miejscu jest komentarz, czyli tekst widoczny dla programisty, ale ignorowany przez interpreter Pythona. Dzięki niemu możemy robić dowolne notatki w swoich programach. Jest to bardzo ważne, bo po dłuższym czasie od napisania programu możemy nie pamiętać, co jest przechowywane w poszczególnych zmiennych czy co robią poszczególne instrukcje; bez komentarzy zorientowanie się nawet we własnym kodzie może być trudne, a bardzo trudne, jeśli pisał go ktoś inny.

Zadania do samodzielnego rozwiązania

Zadanie 1

Napisz programy, w których stworzysz kilka zmiennych. Przypisz im pewne wartości i wyświetl je. Możesz wzorować się na poniższym przykładzie:

```
pierwszy = "kolega"
print(pierwszy)
```

Zadanie 2

Zobacz, które znaki można umieszczać w zmiennych, np.: ?, _, -, +, @ itp. Czy cyfra albo podkreślnik (_) może być na początku nazwy zmiennej?

Zadanie 3

Podaj typy zmiennych:

```
Zmienna = 2.3
Zmienna = 3
Zmienna = 2,3
Zmienna = 'Asia'
Zmienna = "Marysia"
Zmienna = 2,3
```

Czy próba stworzenia którejś ze zmiennych zadeklarowanych w ten sposób spowoduje błąd? Jeśli tak, to której i dlaczego?

Zadanie 4

Jaki będzie wynik poniższych programów? Określ typ danych wyników.

```
# Pierwszy program
w = 4
s = 7
z = w + s

# Drugi program
w = '10'
s = '23'
z = w + s

# Trzeci program
w = 345
s = '90'
z = w + s

# Czwarty program
w = "43"
s = '56'
z = w + s

# Piąty program
w = 23
s = 45.0
z = w + s

# Szósty program
w = 67.0
```

```
s = 45.0
z = w + s

# Siódmy program
w = 23
s = 46.0
z = s / w
```

Zadanie 5

Sprawdź, które z wyrażeń `w1`, `w2`, `w3`, `w4` są prawdziwe, czyli mają wartość `True`, a które fałszywe — mają wartość `False`. Nie przejmuj się, jeśli na razie nie rozumiesz wszystkich użytych w tym kodzie konstrukcji.

```
# Pierwszy program
a = True
b = False
w1 = a and b
w2 = a or b
w3 = not(a) or b
w4 = not(a or b)

# Drugi program
a = ('On' == 'on')
b = (5 > 6 or True)
w1 = a and b
w2 = a or b
w3 = not(a) or b
w4 = not(a or b)
```

Zadanie 6

Poukładaj zdanie ze zmiennych. Połącz i wyświetl.

1. `a = 'Ja', b = 'zakupy', c ='lubię', d = 'bardzo'`
2. `a = 'poniedziałku', b ='lubię', c = 'Nie'`
3. `a= 'Nemo', b = 'jest', c = 'Gdzie'`

Zadanie 7

Wykonanie których spośród poniższych przykładów spowoduje błąd i dlaczego?

1. `a = 5, b = 7, c = a + b`
2. `a = 13, b = -6, c = a + b`
3. `a = 51, b = '7', c = a + b`
4. `a = '26', b = 7, c = a + b`

5. `a = '26', b = 7.7, c = a + b`

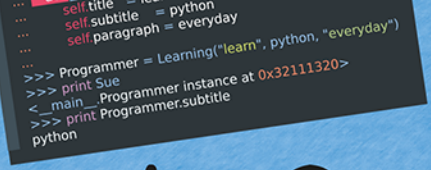
6. `a = '37', b = '7.7', c = a + b`

7. `a = 47.3, b = 7.7, c = a + b`

8. `a = 47.3, b = 7.7, c = a / b`

9. `a = 67.7, b = '7.9', c = a / b`


PROGRAM
PARTNERSKI
GRUPY HELION
1. ZAREJESTRUJ SIĘ
2. PREZENTUJ KSIĄŻKI
3. ZBIERAJ PROWIZJĘ
Zmień swoją stronę WWW w działający bankomat!
Dowiedz się więcej i dołącz już dzisiaj!
http://program-partnerski.helion.pl
GRUPA
Helion



Dlaczego właśnie Python?

Pierwsza odpowiedź, jaka się nasuwa, jest dość oczywista: ponieważ Python to jeden z najprostszych do nauki języków programowania. **Najkrótszy program zajmuje tylko JEDNĄ linijkę.** Z małą pomocą naszego praktycznego zeszytu ćwiczeń każdy adept sztuki programowania zdoła szybko opanować podstawy tego języka i zacznie w nim pisać swoje pierwsze programy.

Druga odpowiedź wydaje się z pozoru zaprzeczać pierwszej. Bo choć Python jest prosty, to drzemią w nim wielkie możliwości. Korzystają z niego bowiem zarówno początkujący programiści, jak i biegli w swoim rzemiośle developerzy aplikacji mobilnych i desktopowych, prowadzi się w nim obliczenia związane ze sztuczną inteligencją, tworzy moduły dla operacji matematycznych, statystycznych oraz finansowych. **Programiści Pythona należą do najlepiej opłacanych specjalistów na świecie!**

Wygląda na to, że przygodę z programowaniem faktycznie warto zacząć od Pythona. **Dzięki pracy z naszym praktycznym zeszytem:**

* *Poznasz software, w którym będziesz działać*
* *Opanujesz podstawowe terminy, których używają programiści Pythona*
* *Nauczysz się wykonywać rozmaite operacje w tym języku*
* *Zaczniesz kodować!*

Helion
helion.pl
HELION SA
ul. Kościuszki 1c
44-100 Gliwice
tel.: 32 230 98 63
helion@helion.pl

KOD KORZYŚCI
Sięgnij po więcej!

ISBN 978-83-283-5820-1
9 788328 358201

INFORMATYKA W NAJLEPSZYM WYDANIU

Cena: 39,90 zł