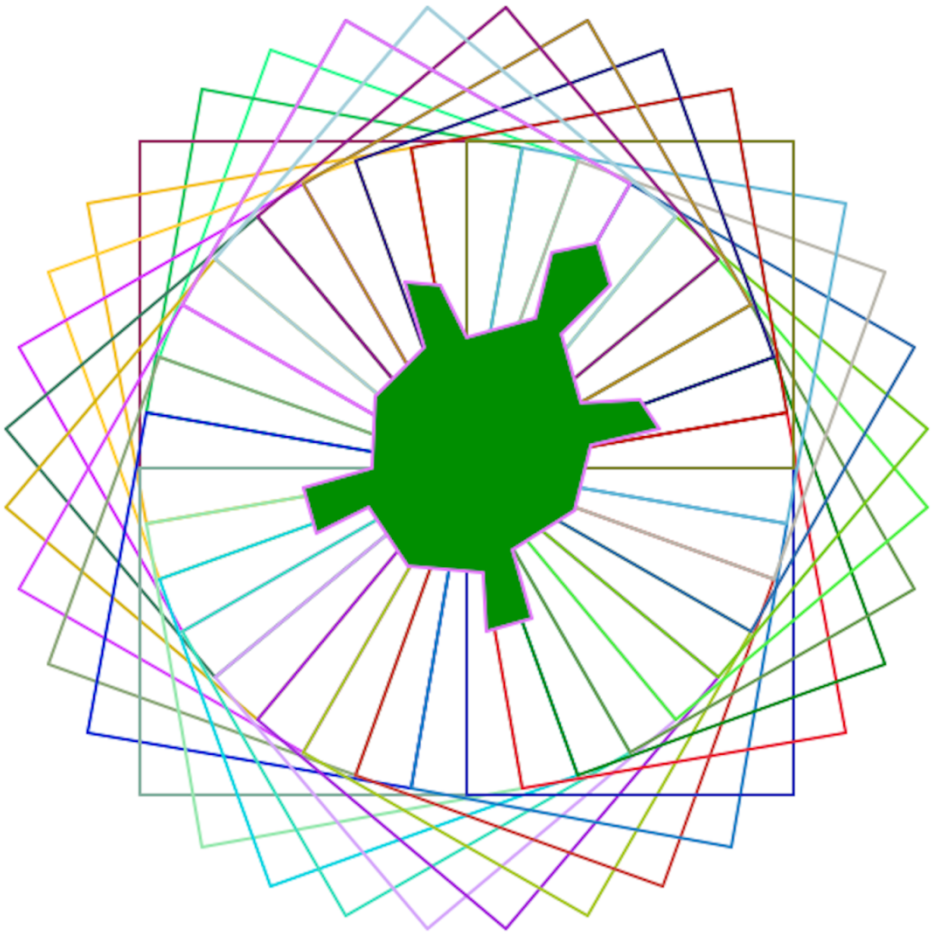
Mariusz Nierzwicki

Python

Wprowadzenie do programowania z wykorzystaniem biblioteki turtle



Projekt okładki: Mariusz Nierzwicki

Mariusz Nierzwicki
www.mclass.pl
e-mail: m.nierzwicki@mclass.pl

ISBN*:* 978-83-964950-0-6

Spis treści

WPROWADZENIE, CZYLI OD CZEGO ZACZĄĆ PROGRAMOWANIE 5

1. GRAFIKA ŻÓŁWIA – PODSTAWOWE POLECENIA ZWIĄZANE Z RUCHEM 6

1.1. RUCH DO PRZODU 6
1.2. RUCH DO TYŁU 6
1.3. SKRĘCANIE W PRAWO 7
1.4. SKRĘCANIE W LEWO 7
1.5. JAK ZATRZYMAĆ DZIAŁANIE PROGRAMU? 8
1.6. ALE GDZIE TEN ŻÓŁW? 8

2. WPROWADZENIE DO ZMIENNYCH - RYSUJEMY KWADRAT 10

3. KOLOR TŁA ORAZ WYBRANE WŁAŚCIWOŚCI PIÓRA 12

3.1. ZMIANA KOLORU TŁA 12
3.2. ZMIANA KOLORU LINII 12
3.3. SZEROKOŚĆ PIÓRA 12
3.4. PODNOSZENIE PIÓRA 13

4. WPROWADZENIE DO PĘTLI FOR 14

5. MODYFIKUJEMY WARTOŚĆ ZMIENNEJ W PĘTLI FOR 15

6. FUNKCJE W JĘZYKU PYTHON 17

7. WYPEŁNIANIE FIGUR WYBRANYM KOLOREM 19

8. PROSTSZE RYSOWANIE OKRĘGU 20

9. FUNKCJA POBIERAJĄCA DANE LICZBOWE OD UŻYTKOWNIKA 22

10. INSTRUKCJA WARUNKOWA IF 24

11. STEROWANIE ŻÓŁWIEM PRZY POMOCY KLAWIATURY 26

12. WPROWADZENIE DO LICZB LOSOWYCH 27

13. WYŚWIETLANIE TEKSTU I LICZB 29

ZADANIA 31

PRZYKŁADOWE ROZWIĄZANIA ZADAŃ: 36

Wstęp

O programowaniu i pracy programistów słyszał dziś zapewne każdy. Wysokie pensje w tym zawodzie czy też marzenia o stworzeniu własnej gry, w wielu przypadkach wpływają na zainteresowanie młodych ludzi nauką programowania. Jednak po początkowej fazie fascynacji i radości z odkrywania nowych rzeczy, często, wraz z pierwszymi napotykanymi trudnościami w zrozumieniu niektórych zagadnień, pojawia się rezygnacja.

Zadaniem niniejszej publikacji jest przybliżenie młodzieży podstaw programowania w jednym z najpopularniejszych obecnie języków programowania – języku Python, w jak najprostszy sposób. Dlaczego akurat ten język? Wybór jest nieprzypadkowy, ponieważ język ten posiada możliwość zaimportowania biblioteki turtle i rozpoczęcia nauki z wykorzystaniem tzw. grafiki żółwia, która pochodzi z lat 60. XX wieku i stosowana była w języku LOGO. Język ten został specjalnie stworzony do nauki programowania, gdzie za pomocą odpowiednich komend sterowało się żółwiem i tworzyło różne wzory graficzne na wirtualnej tablicy. Poznaniem biblioteki turtle zajmiemy się właśnie w tej publikacji, gdzie pojęcia takie jak: zmienna, funkcja, pętla czy instrukcja warunkowa, zostaną przedstawione w jak najbardziej przystępny sposób i będą idealną bazą do dalszego poznawania języka Python.

Należy jednak bowiem pamiętać, że język Python to nie tylko grafika żółwia, jest to pełnoprawny język programowania i stosowany jest w wielu obszarach, takich jak chociażby: analiza danych, tworzenie gier 2D, sztuczna inteligencja, uczenie maszynowe, a nawet serwisy internetowe.

Do każdego rozdziału przypisane są zadania (wraz z przykładowymi rozwiązaniami), gdzie Czytelnik będzie miał możliwość zweryfikowania zdobytej wiedzy i wykorzystania jej w praktyce. Nie da się nauczyć programowania bez aktywnego i samodzielnego kodowania. Tutaj nie ma drogi na skróty.

Życzę miłej lektury i wytrwałości w rozwiązywaniu zadań. Powodzenia.

Wprowadzenie, czyli od czego zacząć programowanie

Zanim przystąpimy do właściwej nauki programowania i zaczniemy tworzyć pierwsze programy, na początku niezbędna jest instalacja samego środowiska Python. Aktualną wersję do pobrania i zainstalowania znaleźć można na stronie **https://www.python.org/downloads/**

Mimo, że strona ta jest w języku angielskim, nie martw się jeśli nie rozumiesz jeszcze tego języka, po prostu klikamy na niej na duży żółty przycisk z napisem **„Download Python"** z numerem wersji. Po pobraniu pliku instalacyjnego, uruchamiamy go i rozpoczynamy instalację. Należy tylko pamiętać aby zaznaczyć opcję „**Add Python to PATH**", a dopiero później kliknąć na **„Install Now"**. Dalszy proces instalacji jest już bardzo intuicyjny.

Sama instalacja środowiska Python to nie wszystko. Potrzebne będzie jeszcze tak zwane IDE (z języka angielskiego - ***Integrated Development Environment***) czyli zintegrowane środowisko programistyczne, w którym będziemy pisać nasze programy.

Do najpopularniejszych IDE Pythona zaliczyć na pewno można:

- Visual Studio Code,
- PyCharm.

Są to jednak bardzo rozbudowane środowiska, o dużych możliwościach, używane na co dzień przez zawodowych programistów. Nic nie stoi na przeszkodzie by zacząć korzystać z któregoś z nich, jednakże na początek przygody z programowaniem polecam zainstalowanie programu **Thonny** – IDE dla języka Python, przeznaczonego dla początkujących. Program ten znaleźć można na stronie: **https://thonny.org/**

Instalacja tego IDE jest bardzo prosta – na stronie w ramce „**Download**", klikamy na nazwę systemu operacyjnego z którego korzystamy i po pobraniu pliku instalacyjnego, uruchamiamy go.

Dopiero teraz po zainstalowaniu IDE, wyposażeni w odpowiednie narzędzia, możemy przejść do lektury pierwszego rozdziału i praktycznej nauki programowania.

1. Grafika żółwia – podstawowe polecenia związane z ruchem

Przygodę z programowaniem w języku Python rozpoczniemy od poznania podstawowych poleceń sterowania żółwiem – ruch do przodu, ruch do tyłu, skręcanie w prawo, skręcanie w lewo. Zanim jednak stworzymy pierwszy program, należy pamiętać, aby na początku kodu zaimportować (czyli dodać) moduł umożliwiający, przy pomocy tak zwanego żółwia, tworzenie różnych obrazów i kształtów na wirtualnej tablicy.

Pisanie kodu programu z grafiką żółwia zaczniemy zatem od polecenia: **import turtle**.

1.1. Ruch do przodu

Aby żółw wykonał ruch przed siebie zastosujemy polecenie **turtle.forward()** lub w skrócie **turtle.fd()** gdzie w nawiasie wpisać należy liczbę pikseli (kroków), które żółw ma przejść. W języku angielskim ***forward*** znaczy ***naprzód***.

W poniższym przykładzie, żółw wykona 100 kroków (przejdzie odległość 100 pikseli), a ponieważ domyślnie zostawia za sobą ślad, efektem będzie narysowana linia prosta o długości 100 pikseli.

Domyślnie żółw „patrzy” w kierunku prawej krawędzi ekranu.

```
1:  import turtle
2:  turtle.forward(100)
```

Listing 1.1

1.2. Ruch do tyłu

Nasz żółw może również cofać się, czyli wykonywać ruch do tyłu. Służy do tego polecenie **turtle.backward()** lub w skrócie **turtle.bk()** gdzie w nawiasie wpisać należy liczbę pikseli (kroków), które żółw ma przejść. W języku angielskim ***backward*** znaczy ***do tyłu***.

W poniższym przykładzie, żółw wykona 100 kroków do tyłu, zostawiając za sobą narysowaną linię.

```
1:  import turtle
2:  turtle.backward(100)
```

Listing 1.2

1.3. Skręcanie w prawo

Poznaliśmy już polecenia pozwalające na ruch do przodu i ruch do tyłu, zatem pora na polecenia pozwalające skręcać naszym żółwiem. Zaczniemy od obrotu w prawo. Użyjemy w tym celu polecenia **turtle.right()** lub w skrócie **turtle.rt()** w nawiasie wpisując miarę kąta pod jakim żółw ma skręcić. W języku angielskim ***right*** znaczy ***prawo***.

W poniższym przykładzie, żółw pójdzie przed siebie przez odległość 100 pikseli (zostawiając za sobą ślad), a następnie obróci się **w swoją prawą stronę pod kątem 90 stopni**, czyli obróci się w prawo.

```
1:  import turtle
2:  turtle.forward(100)
3:  turtle.right(90)
```

Listing 1.3

1.4. Skręcanie w lewo

Aby skręcić żółwiem w lewą stronę, jak już pewnie się domyślasz, użyjemy polecenia **turtle.left()** lub w skrócie **turtle.lt()** wpisując w nawiasie miarę kąta. W języku angielskim ***left*** znaczy ***lewo***.

W poniższym przykładzie, żółw pójdzie przed siebie przez odległość 100 pikseli (zostawiając za sobą ślad), a następnie obróci się **w swoją lewą stronę pod kątem 90 stopni**, czyli obróci się w lewo.

```
1:  import turtle
2:  turtle.forward(100)
3:  turtle.left(90)
```

Listing 1.4

UWAGA: Aby skręcić w lewo pod kątem 90 stopni najprościej będzie użyć polecenia **turtle.left(90)** ale gdybyśmy użyli polecenia **turtle.right(270)** efekt byłby identyczny. Przetestuj poniższe dwa kody programu (różnica jest tylko w 3 linijce kodu) i spróbuj odpowiedzieć, który kod działa szybciej?

```
1:  import turtle
2:  turtle.forward(100)
3:  turtle.left(90)
4:  turtle.forward(100)
```

```
1:  import turtle
2:  turtle.forward(100)
3:  turtle.right(270)
4:  turtle.forward(100)
```

Listing 1.4.1

1.5. Jak zatrzymać działanie programu?

W czasie testowania powyższych przykładów zapewne dało się zauważyć, że okienko z grafiką żółwia po wykonaniu zaprogramowanych poleceń szybko się zamykało (program się kończył). Zazwyczaj będziemy chcieli popatrzeć na efekty naszych programów odrobinę dłużej i w tym celu powinniśmy skorzystać z polecenia: **turtle.exitonclick()**.

Jeśli dodamy to polecenie **na końcu kodu programu**, to po uruchomieniu i narysowaniu przez żółwia określonych wzorów, będzie on tak długo włączony, aż klikniemy myszką na oknie z grafiką żółwia. Z języka angielskiego zwrot ***exit on click*** znaczy po prostu ***wyjdź po kliknięciu***.

```
1:  import turtle
2:  turtle.forward(100)
3:
4:  turtle.exitonclick()
```

Listing 1.5

1.6. Ale gdzie ten żółw?

Po uruchomieniu pierwszych skryptów, zastanawiasz się pewnie o jakim żółwiu jest mowa w tej publikacji, przecież nie widać żadnego żółwia tylko przemieszczającą się strzałkę. Masz rację. W Pythonie, w module turtle, domyślnym obiektem, którym sterujemy, jest właśnie strzałka (a właściwe grot strzałki), ale można to w prosty sposób zmienić. Służy do tego polecenie: **turtle.shape()**, gdzie w nawiasie wpisujemy w cudzysłowie jedną z poniższych opcji:

- turtle,
- arrow,
- circle,

- square,
- triangle,
- classic.

Na początek przygody z programowaniem z grafiką żółwia, proponuję wybrać kształt (obiekt), który jednoznacznie pozwoli na zaobserwowanie kierunku obrotu. Ułatwi to z pewnością ewentualne poprawianie kodu. W przypadku kwadratu (square), skręt będzie bardzo trudno zauważyć, a w przypadku koła (circle) będzie to wręcz niemożliwe.

Widoczność obiektu rysującego można też dowolnie, w zależności od potrzeb, włączać i wyłączać (pokazywać i ukrywać). Służą do tego następujące polecenia:

- **turtle.hideturtle()** – ukrywa żółwia. Z języka angielskiego ***hide*** znaczy ***ukryć***.
- **turtle.showturtle()** – ustawia żółwia widocznym. Domyślnie żółw ma ustawioną tą właśnie opcję, dlatego nie musimy w każdym programie dodawać tego polecenia gdy chcemy aby podczas rysowania (działania programu) żółw (czy też inny kształt rysujący) był widoczny. Z języka angielskiego ***show*** znaczy ***pokazać***.

Można również w tym przypadku wykorzystać skrócone polecenia:

- **turtle.ht()** – zamiast turtle.hideturtle(), oraz
- **turtle.st()** – zamiast turtle.show().

Poniższy przykład demonstruje użycie wybranych, wyżej opisanych, poleceń. Puste linie 3, 7 i 8 służą tylko temu aby kod programu był czytelny i przejrzysty.

```
1:  import turtle
2:  turtle.shape("turtle")
3:
4:  turtle.forward(200)
5:  turtle.left(90)
6:  turtle.forward(200)
7:
8:  turtle.hideturtle()
9:
10: turtle.exitonclick()
```

Listing 1.6

Jeśli chcesz sprawdzić swoją dotychczasową wiedzę, przejdź do rozdziału **ZADANIA** i spróbuj samodzielnie rozwiązać **Zadanie 1**.

2. Wprowadzenie do zmiennych - rysujemy kwadrat

W rozdziale drugim poznamy czym są tak zwane zmienne, ale nim to nastąpi wykonajmy proste zadanie, które posłuży nam jako baza do wytłumaczenia tego zagadnienia.

Zadanie polega na zaprogramowaniu żółwia w taki sposób aby narysował kwadrat o boku równym 100 pikseli.

Bazując na wiadomościach z poprzedniego rozdziału, kod Twojego programu będzie zapewne wyglądać tak jak w poniższym przykładzie:

```
1:  import turtle
2:
3:  turtle.forward(100)
4:  turtle.left(90)
5:  turtle.forward(100)
6:  turtle.left(90)
7:  turtle.forward(100)
8:  turtle.left(90)
9:  turtle.forward(100)
10: turtle.left(90)
11:
12: turtle.exitonclick()
```

Listing 2.1

Zauważ, że za każdym razem, w poleceniu **turtle.forward()** w nawiasie wpisywaliśmy wartość 100. W przypadku, kiedy chcielibyśmy zmienić wielkość kwadratu (długość jego boku), cztery razy musielibyśmy zastępować 100 nową wartością. W tym momencie prościej byłoby posłużyć się zmienną, czyli konstrukcją, gdzie pod ustaloną przez nas nazwą będzie przechowywana w pamięci komputera określona przez nas wartość (wartość ta oczywiście może zmieniać się w trakcie działania programu, stąd nazwa zmienna).

W przypadku zastosowania zmiennej, gdybyśmy chcieli dokonać modyfikacji programu i zmienić długość boku, to zmiany dokonać musielibyśmy tylko w jednym miejscu. Musielibyśmy tylko zmienić wartość zmiennej, zakładając, że już podczas pisania programu w poleceniu turtle.forward() w nawiasie zamiast liczby 100, wpisalibyśmy nazwę zmiennej (zobacz Listing 2.2).

W tym momencie warto wspomnieć o nazwach zmiennych. Warto stosować takie nazwy, które jednoznacznie będą wskazywać konkretnie co to za zmienna i jakie wartości przechowuje. Pamiętać należy również o tym, że w nazwach zmiennych nie możemy stosować spacji, stąd trzy konwencje stosowania nazw zmiennych (trzy sposoby tworzenia nazw zmiennych):

- snake_case

- camelCase
- PascalCase.

Gdybyśmy chcieli zmienną nazwać po prostu „bok a” to użylibyśmy nazw:

- **bok_a** (w przypadku konwencji snake_case, spacje zastępujemy znakiem podkreślenia _) lub,
- **bokA** (w przypadku konwencji camelCase, gdzie **każdy kolejny** wyraz rozpoczynamy wielką literą) lub,
- **BokA** (w przypadku konwencji PascalCase, gdzie **każdy** wyraz rozpoczynamy wielką literą).

W poniższym przykładzie posłużono się konwencją snake_case.

```
1:  import turtle
2:
3:  bok_a = 100
4:
5:  turtle.forward(bok_a)
6:  turtle.left(90)
7:  turtle.forward(bok_a)
8:  turtle.left(90)
9:  turtle.forward(bok_a)
10: turtle.left(90)
11: turtle.forward(bok_a)
12: turtle.left(90)
13:
14: turtle.exitonclick()
```

Listing 2.2

W powyższym przykładzie widzimy wprowadzoną **zmienną bok_a**, której przypisano wartość 100 (3 linia kodu). Następnie w poleceniach turtle.forward() w nawiasie wprowadzono tylko nazwę zmiennej. W ten sposób program sam „pobiera” aktualną wartość zmiennej bok_a. W tym przykładzie przez cały czas działania programu wartość zmiennej bok_a nie ulega zmianie i wynosi 100.

W kolejnych rozdziałach poznamy polecenia pozwalające pobrać wartość zmiennej od użytkownika programu i stworzymy program, który zanim zacznie rysować kwadrat, zapyta najpierw użytkownika o długość boku kwadratu, który ma zostać narysowany.

W rozdziale **ZADANIA** znajdziesz **Zadanie 2**, które wykonane samodzielnie pozwoli lepiej zrozumieć zagadnienie zmiennych.

3. Kolor tła oraz wybrane właściwości pióra

W niniejszym rozdziale poznamy kilka poleceń modyfikujących domyślne wartości koloru tła czy koloru i wielkości pióra, którym żółw rysuje.

3.1. Zmiana koloru tła

Aby zmienić kolor tła naszego okienka (wirtualnej tablicy), w którym porusza się i rysuje żółw, musimy użyć następującego polecenia: **turtle.bgcolor()** w nawiasie, w cudzysłowie, wpisując w języku angielskim nazwę koloru którego chcemy użyć lub tzw. kod koloru (np. #07538d). Tabele z kodami kolorów bez problemów znaleźć można w Internecie.

Trzecią opcją jest użycie palety barw RGB, wówczas w nawiasie wpisujemy trzy wartości liczbowe od 0 do 255, oddzielone przecinkiem. Skrót RGB pochodzi od pierwszych liter angielskich nazw kolorów *red*, *green*, *blue* czyli trzech podstawowych kolorów, przy pomocy których, tak jak malarz mieszający farby, możemy stworzyć dowolną barwę.

Na przykład zapis:

```
turtle.bgcolor(255, 0, 0)
```

oznacza użycie koloru czerwonego, ponieważ parametr *red* ma maksymalną wartość 255, zaś *green* i *blue* ma wartość 0, czyli całkowity brak tych barw w naszym kolorze. Tabele z kodami kolorów w formacie RGB również bez większych trudności znaleźć można w Internecie.

W przypadku użycia palety barw RGB koniecznym jest dodanie *(*najlepiej zaraz na początku naszego programu, tuż po zaimportowaniu biblioteki turtle) następującej linii kodu*:*

```
turtle.colormode(255)
```

Skrót **bgcolor** pochodzi z angielskiego określenia ***background color*****,** czyli dosłownie, ***kolor tła***.

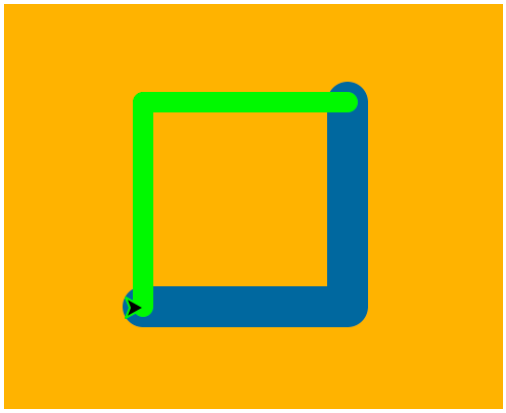
3.2. Zmiana koloru linii

Aby zmienić kolor linii, którą zostawia za sobą żółw musimy użyć polecenia: **turtle.color()** lub **turtle.pencolor()** w nawiasie, w cudzysłowie wpisując w języku angielskim nazwę koloru którego chcemy użyć lub tzw. kod koloru lub palety barw RGB (identycznie jak w przypadku zmiany koloru tła).

3.3. Szerokość pióra

Aby zmienić wielkość pióra, którym żółw maluje, musimy skorzystać z polecenia **turtle.pensize()** i w nawiasie wpisać wartość liczbową (bez przecinka). Liczba ta określa szerokość pióra w pikselach.

```
1:  import turtle
2:  turtle.bgcolor("orange")
3:  turtle.pencolor("#07538d")
4:  turtle.pensize(20)
5:
6:  turtle.forward(100)
7:  turtle.left(90)
8:  turtle.forward(100)
9:  turtle.left(90)
10:
11: turtle.pensize(10)
12: turtle.pencolor("green")
13:
14: turtle.forward(100)
15: turtle.left(90)
16: turtle.forward(100)
17: turtle.left(90)
18:
19: turtle.exitonclick()
```



Listing 3.3. i efekt jego działania

W powyższym przykładzie zmieniono kolor tła na pomarańczowy (z jęz. angielskiego ***orange***), zmieniono również kolor pióra i ustawiono jego wielkość na 20. Następnie zaprogramowano żółwia tak aby rysował dwa boki kwadratu, po czym ponownie zmieniono wielkość pióra (na 10) i jego kolor ustawiono na zielony (z jęz. angielskiego ***green***). Na koniec żółw rysował kolejne dwa boki kwadratu.

W rozdziale **ZADANIA** znajdziesz **Zadanie 3.1**, które, wykonane samodzielnie, pozwoli lepiej zrozumieć poznane w tym podrozdziale zagadnienia.

3.4. Podnoszenie pióra

Tak jak już wcześniej wspomniano, domyślnie, żółw przemieszczając się zostawia za sobą ślad (rysuje). W niektórych projektach z pewnością zaistnieje potrzeba, aby żółw przemieścił się w taki sposób, aby nie zostawiał za sobą śladu.

Wówczas, aby unieść pióro, należy użyć polecenia: **turtle.penup()** lub w skrócie **turtle.pu()**. Z języka angielskiego, zwrot ***pen up*** znaczy po prostu ***pióro w górę***.

Aby ponownie żółw opuścił pióro i ponownie rysował przemieszczając się, należy użyć polecenia: **turtle. pendown()** lub w skrócie **turtle.pd().** Z języka angielskiego, zwrot ***pen down*** znaczy po prostu ***pióro w dół***.

W rozdziale **ZADANIA** znajdziesz **Zadanie 3.2**, które wykonane samodzielnie pozwoli lepiej zrozumieć poznane w tym podrozdziale zagadnienie.

4. Wprowadzenie do pętli for

W jednym z poprzednich przykładów programowaliśmy żółwia tak aby narysował kwadrat o boku 100 pikseli (Listing 2.1). Jak pewnie dostrzegliście, tak naprawdę powtarzaliśmy cztery razy te same czynności:

- ruch do przodu (100 pikseli)
- skręt w lewo.

W językach programowania do powtarzania pewnych fragmentów kodu programu służy, między innymi, tak zwana pętla **for**.

W języku Python składnia pętli for wygląda następująco:

```
for nazwa zmiennej in range(ilość powtórzeń):
    operacje działające w pętli
```

W naszych przykładach za nazwę zmiennej możemy podstawić dowolną nazwę na przykład x, natomiast w miejsce ilość powtórzeń wpisujemy liczbę całkowitą, określającą, ile razy operacje w pętli for mają zostać powtórzone.

UWAGA: W języku Python istotna jest składnia kodu. Do tej pory, w niniejszej publikacji, wszystkie nowe polecenia wpisywaliśmy od razu, bezpośrednio na początku każdej nowej linii kodu. Pętla for jest pierwszym poleceniem, gdzie o ile początek pętli ze słowem for rozpoczynamy od początku nowej linii, o tyle polecenia, które w tej pętli mają działać (mają zostać powtórzone) rozpoczynamy również każde w nowej linii, ale na początku wciskamy na klawiaturze klawisz **TAB** (robimy tak zwane wcięcie tabulatorem).

Wynika to z tego, że w języku Python nie ma symbolu kończącego pętlę tak jak na przykład w języku Java, gdzie operacje działające w pętli zapisane są między nawiasami { }.

Posiadając już niezbędną wiedzę, stwórzmy prosty program rysujący kwadrat o boku 100, ale wykorzystujący już pętlę for.

```
1: import turtle
2:
3: for x in range(4):
4:     turtle.forward(100)
5:     turtle.left(90)
6:
7: turtle.exitonclick()
```

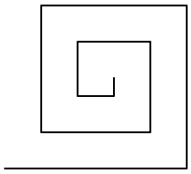
Listing 4

Jak można zauważyć kod programu jest krótszy niż ten z rozdziału 2. W tym przypadku „zaoszczędziliśmy" tylko pięć linii kodu, jednak „zysk" będzie tym większy, im więcej poleceń zechcemy powtarzać.

W rozdziale **ZADANIA** znajdziesz **Zadanie 4**, które wykonane samodzielnie pozwoli lepiej zrozumieć pętlę for.

5. Modyfikujemy wartość zmiennej w pętli *for*

W poprzednich rozdziałach poznaliśmy już czym są zmienne i w jaki sposób korzystamy z pętli for. W tym rozdziale połączymy tą wiedzę i stworzymy program, którego zadaniem będzie rysowanie spirali – labiryntu, jak na poniższym zdjęciu.



Zauważ, że nasza spirala składa się z 10 linii o różnej długości. Pierwsza linia (położona najniżej) ma długość 100 pikseli, a każda kolejna jest krótsza od poprzedniej o 10 pikseli.

Jak już zapewne się domyślasz, skorzystamy tutaj z pętli for w której powtórzymy 10 razy (ponieważ chcemy 10 linii) dwie czynności – rysowanie linii i skręcanie w lewo pod kątem 90 stopni.

Jedyną rzeczą, która na początku wydawać mogłaby się dużym problemem, jest zmieniająca się długość linii. Posłużymy się tutaj zmienną, którą nazwiemy **dlugosc_linii** i na początku przypiszemy jej wartość 100. Następnie w pętli, po poleceniach rysowania i skręcania, będziemy pomniejszać wartość naszej zmiennej o 10.

Zapis:

dlugosc_linii = dlugosc_linii – 10

oznacza po prostu, że zmienna **dlugosc_linii** wynosić będzie: dotychczasowa wartość tej zmiennej minus 10.

Kod naszego programu powinien wyglądać tak jak na poniższym listingu.

```
1:  import turtle
2:
3:  dlugosc_linii = 100
4:
5:  for x in range(10):
6:      turtle.forward(dlugosc_linii)
7:      turtle.left(90)
8:      dlugosc_linii = dlugosc_linii - 10
9:
10: turtle.exitonclick()
```

Listing 5

W pierwszym przejściu pętli for (pierwsze z dziesięciu powtórzeń) nasz żółw przemieszcza się do przodu o długość przypisaną zmiennej **dlugosc_linni** (3 linia kodu, w tym przypadku 100, bo taką wartość przypisaliśmy tej zmiennej jeszcze przed pętlą for), następnie skręca w lewo pod kątem 90 stopni i zmiennej **dlugosc_linii** przypisuje nową wartość: aktualna wartość zmiennej **dlugosc_linii** minus 10. Zatem wartość zmiennej na koniec pierwszego przejścia pętli for wynosić będzie 90.

W drugim przejściu pętli for (drugie z dziesięciu powtórzeń) nasz żółw ponownie przemieszcza się przed siebie na odległość równą aktualnej wartości zmiennej **dlugosc_linii** czyli tym razem 90. Następnie skręca w lewo pod kątem 90 stopni i pomniejsza kolejny raz wartość zmiennej **dlugosc_linii** o 10. Zatem wartość tej zmiennej na koniec drugiego przejścia pętli for wynosić będzie już 80.

W kolejnych przejściach pętli for, wszystko znowu się powtarza, tak, że w ostatnim 10 przejściu, wartość zmiennej **dlugosc_linii** wynosi już 10 pikseli.

W linijce 8 można by też użyć skróconego zapisu:

```
8:  dlugosc_linii -= 10
```

Spróbuj samodzielnie zmodyfikować fragmenty kodu tego programu, np. wartość zmiennej **dlugosc_linii**, kąt skrętu czy ilość powtórzeń pętli for.

W rozdziale **ZADANIA** znajdziesz **Zadanie 5**, które wykonane samodzielnie pozwoli lepiej zrozumieć poznane zagadnienie.

6. Funkcje w języku Python

Wydzielone fragmenty kodu, które możemy użyć w innym miejscu to tak zwane funkcje. Ich konstrukcja jest bardzo prosta. Używamy tutaj polecenia **def** następnie spacja i nazwa funkcji z nawiasem otwartym i zamkniętym, a na końcu dwukropek.

def nazwafunkcji():

W funkcjach, podobnie jak w przypadku pętli for, każdą nową linię z poleceniami wchodzącymi w skład funkcji, rozpoczynamy od naciśnięcia klawisza **TAB**.

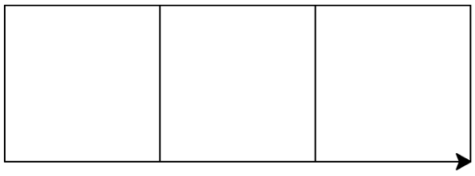
Kiedy chcemy wywołać wykonanie funkcji (uruchomić ją) wystarczy napisać w odpowiednim miejscu naszego programu nazwę funkcji z nawiasem otwartym i zamkniętym (**bez znaku dwukropka na końcu**).

Poniżej prosty przykład z zastosowaniem funkcji – program, który rysuje 3 kwadraty o boku 100, obok siebie.

```
1:  import turtle
2:
3:  def rysujKwadrat():
4:      turtle.forward(100)
5:      turtle.left(90)
6:      turtle.forward(100)
7:      turtle.left(90)
8:      turtle.forward(100)
9:      turtle.left(90)
10:     turtle.forward(100)
11:     turtle.left(90)
12:
13: rysujKwadrat()
14: turtle.forward(100)
15: rysujKwadrat()
16: turtle.forward(100)
17: rysujKwadrat()
18: turtle.forward(100)
19:
20: turtle.exitonclick()
```

Listing 6.1

Jak widzicie, na początku stworzyliśmy funkcję o nazwie **rysujKwadrat()** w której zawarliśmy polecenia odpowiedzialne za rysowanie kwadratu o boku 100. W przykładzie świadomie zrezygnowano z pętli for. Następnie wywołujemy tą funkcję (linia 13), przemieszczamy żółwia o 100 pikseli do przodu i ponawiamy te czynności jeszcze dwa razy. Efekt działania widzimy poniżej:



Oczywiście poznaliśmy już pętlę for, zatem powyższy przykład programu moglibyśmy jeszcze bardziej uprościć poprzez zastosowanie pętli for w funkcji oraz drugiej pętli for, która trzy razy wywoła funkcję **rysujKwadrat().**

Poniżej kod programu z zastosowaniem dwóch pętli for:

```
1:  import turtle
2:
3:  def rysujKwadrat():
4:      for x in range(4):
5:          turtle.forward(100)
6:          turtle.left(90)
7:
8:  for x in range(3):
9:      rysujKwadrat()
10:     turtle.forward(100)
11:
12:  turtle.exitonclick()
```

Listing 6.2

Prawda, że kod programu teraz jest znacznie krótszy i bardziej przejrzysty?

W rozdziale **ZADANIA** znajdziesz **Zadanie 6.1** oraz **Zadanie 6.2**, które wykonane samodzielnie pozwolą lepiej zrozumieć zagadnienie funkcji.

7. Wypełnianie figur wybranym kolorem

Do tej pory jedynie programowaliśmy żółwia, aby rysował określone figury. Pora abyśmy nauczyli się wypełniać je określonym kolorem. Spróbujmy zatem napisać kod programu, którego zadaniem będzie rysowanie kwadratu wypełnionego wybranym kolorem.

Na początek określić musimy jakim kolorem chcemy wypełnić naszą figurę. Służy do tego polecenie:

turtle.fillcolor()

gdzie w nawiasie wpisujemy w cudzysłowie nazwę koloru w języku angielskim lub korzystamy z tzw. kodów kolorów.
Następnie, jeszcze przed rysowaniem wybranej przez nas figury, korzystamy z polecenia:

turtle.begin_fill()

Czyli po prostu rozpocznij wypełnianie.

Po narysowaniu zaprogramowanej i wypełnionej kolorem, figury płaskiej, należy zakończyć wypełnianie, posługując się poleceniem:

turtle.end_fill()

inaczej każda kolejna figura, którą będziemy rysować będzie wypełniania ustawionym przez nas wcześniej kolorem.

Słowo ***fill*** w języku angielskim znaczy po prostu ***wypełnić***, zatem ***begin fill*** oznacza ***rozpocznij wypełnianie***, z kolei ***end fill*** – ***zakończ wypełnianie***.

Zatem kod programu rysującego wypełniony na czarno kwadrat o boku 100, będzie wyglądał następująco:

```
1:  import turtle
2:
3:  turtle.fillcolor("black")
4:
5:  turtle.begin_fill()
6:
7:  for x in range(4):
8:      turtle.forward(100)
9:      turtle.left(90)
10:
```

```
11: turtle.end_fill()
12:
13: turtle.exitonclick()
```

Listing 7

Pamiętaj, że kolory wypełnienia można zmieniać i każdą kolejną figurę, rysowaną w tym samym programie, wypełniać można innym kolorem.

W rozdziale **ZADANIA** znajdziesz **Zadanie 7**, które wykonane samodzielnie pozwoli lepiej zrozumieć poznane zagadnienie.

8. Prostsze rysowanie okręgu

Do tej pory (również w zadaniach) programowaliśmy rysowanie poszczególnych figur takich jak: kwadrat, prostokąt, trójkąt czy ośmiokąt. Gdybyśmy chcieli narysować okrąg wystarczyłoby napisać program jak na poniższym listingu.

```
1: import turtle
2:
3: for x in range(360):
4:     turtle.fd(1)
5:     turtle.lt(1)
6:
7: turtle.exitonclick()
```

Listing 8.1

Jak widać, w pętli, 360 razy powtórzylibyśmy ruch do przodu o 1 piksel i skręt w lewo o 1 stopień.

Przepisując i uruchamiając powyższy kod samemu dojść można do wniosku, że ten sposób rysowania okręgu nie jest zbyt wygodny. Na szczęście, autorzy biblioteki Python turtle dodali prostszy sposób na rysowanie okręgu. Służy temu polecenie:

turtle.circle()

gdzie w nawiasie wpisujemy obowiązkowo jeden parametr, a mianowicie długość promienia wyrażoną w pikselach. Program na poniższym listingu, rysuje okrąg o promieniu 50 pikseli.

```
1: import turtle
2:
3: turtle.circle(50)
4:
5: turtle.exitonclick()
```

Listing 8.2

Wśród parametrów metody **turtle.circle()** oprócz obowiązkowego promienia, możemy dodać jeszcze informację jaką część okręgu chcemy narysować (wyrażamy tą wartość w stopniach). Cały okrąg ma 360 stopni, zatem gdybyśmy chcieli narysować tylko pół okręgu, to właśnie ten drugi parametr powinien wynosić 180.

Trzeci ostatni, również opcjonalny parametr, służy do określenia ilości kroków do narysowania okręgu. Parametry oddzielamy przecinkiem.

Poniżej dwa przykłady, które obrazują wpływ ostatniego parametru na efekt końcowy.

```
1: import turtle
2:
3: turtle.circle(50,180,4)
4:
5: turtle.exitonclick()
```

```
1: import turtle
2:
3: turtle.circle(50,180,180)
4:
5: turtle.exitonclick()
```

Listing 8.3

W rozdziale **ZADANIA** znajdziesz **Zadanie 8**, które wykonane samodzielnie pozwoli lepiej zrozumieć poznane w tym rozdziale zagadnienie.

9. Funkcja pobierająca dane liczbowe od użytkownika

W rozdziale drugim, w którym była mowa o zmiennych, wspomniano o możliwości wprowadzania wartości zmiennych przez użytkownika, już po uruchomieniu programu. Do pobierania od użytkownika wartości liczbowych, bo takimi w tym momencie się zajmiemy, służy polecenie:

turtle.numinput()

gdzie w nawiasie wprowadzić musimy dwa parametry obowiązkowe i możemy dodać trzy opcjonalne.

Do parametrów obowiązkowych zalicza się tytuł wyskakującego okienka (pisany w cudzysłowie) oraz wyświetlane w nim pytanie, na które oczekujemy odpowiedzi od użytkownika programu. Parametr z pytaniem również zapisujemy w cudzysłowie. Parametry oddzielamy przecinkiem.

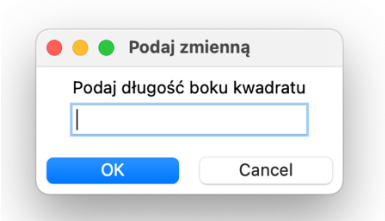
Wyjaśnijmy jeszcze co oznacza słowo **numinput**. „Num" to skrót od angielskiego słowa ***number*** czyli po prostu ***numer***, z kolei ***input*** znaczy ***wejście***.

Stwórzmy wspólnie prosty program pobierający od użytkownika długość boku kwadratu, który ma zostać narysowany.

```
1:  import turtle
2:
3:  bok_a = turtle.numinput("Podaj zmienną", "Podaj długość boku
    kwadratu:")
4:
5:  for x in range(4):
6:      turtle.forward(bok_a)
7:      turtle.left(90)
8:
9:  turtle.exitonclick()
```

Listing 9.1

Zmiennej **bok_a** przypisujemy wartość wprowadzoną przez użytkownika w wyskakującym okienku, które zatytułowane jest ***„Podaj zmienną"*** i zawiera pytanie ***„Podaj długość boku kwadratu"*** (3 linia kodu na powyższym listingu). Następnie w pętli for cztery razy powtarzane są dwie czynności: ruch do przodu o wartość zmiennej **bok_a** (czyli tyle, ile podał użytkownik w wyskakującym okienku) i skręt w lewo pod kątem 90 stopni.



Powyższy przykład zawiera tylko parametry obowiązkowe. Rozważmy jednak sytuację, w której użytkownik podaje na przykład dużą liczbę i kwadrat o takim boku nie zmieściłby się na ekranie. Jeśli nie chcielibyśmy dopuścić do takiej sytuacji, z pomocą przyjdą nam tutaj parametry dodatkowe, gdzie możemy określić wartość domyślną, wartość minimalną i wartość maksymalną.

Chcąc skorzystać z parametrów dodatkowych, w funkcji numinput za parametrami obowiązkowymi możemy dopisać:

- **default=** i liczbę – stworzymy wówczas podpowiedź i podana przez nas liczba pojawi się domyślnie w wyskakującym okienku,
- **minval=** i liczbę – określimy wówczas jaka wartość może być najmniejszą wartością wpisaną przez użytkownika,
- **maxval=** i liczbę – podobnie jak wcześniej, tyle, że tym razem określamy wartość maksymalną jaką może wprowadzić użytkownik.

Z języka angielskiego słowo ***default*** znaczy ***domyślny***. Z kolei **minval** to skrót od angielskiego ***minimum value***, czyli dosłownie ***wartość minimalna***. Podobne sprawa wygląda z **maxval**, które jest skrótem od angielskiego ***maximum value*** i znaczy dosłownie ***wartość maksymalna***.

Spróbuj samodzielnie uzupełnić poprzedni kod programu (3 linia) o parametry opcjonalne, tak jak na poniższym listingu.

```
3: bok_a = turtle.numinput("Podaj zmienną", "Podaj długość boku
             kwadratu:", default=99, minval=10, maxval=150)
```

Listing 9.2

Przetestuj działanie programu modyfikując parametry funkcji numinput() oraz sprawdź co się stanie, już po uruchomieniu programu, jeśli podasz w wyskakującym okienku wartość większą niż przewidziana wartość maksymalna lub minimalna.

W rozdziale **ZADANIA** znajdziesz **Zadanie 9**, które pozwoli wykorzystać wiedzę na temat funkcji numinput() w praktyce. Zadanie to jest nieco trudniejsze, dlatego koniecznie zapoznaj się podpowiedzią.

10. Instrukcja warunkowa *if*

We wszystkich językach programowania bardzo ważnym i często używanym elementem jest instrukcja warunkowa if. Pozwala ona na wykonanie różnych instrukcji w programie w zależności od tego czy dany warunek został spełniony.

Instrukcję warunkową **if** omówimy na prostym przykładzie. Mianowicie stworzymy program, który będzie rysował kwadrat lub okrąg, w zależności od tego, jaką cyfrę użytkownik wpisze w wyskakującym okienku numinput(), które poznaliśmy w poprzednim rozdziale.

Słówko ***if*** z języka angielskiego znaczy dosłownie ***jeśli, jeżeli.***

Składnia tego polecenia wygląda następująco:

```
if (warunek):
    Polecenia, które mają się wykonać jeśli warunek został spełniony
```

Polecenie **if** z warunkiem wpisujemy od razu na początku linii. Z kolei polecenia, które mają się wykonać jeśli warunek będzie spełniony, piszemy z wcięciem robionym tabulatorem (podobnie jak w przypadku funkcji czy pętli for).

W instrukcji warunkowej, oprócz samego polecenia **if** występują jeszcze polecenia **elif** (skrót od else if) i **else**. Z języka angielskiego słówko ***else*** oznacza ***w przeciwnym razie***.

Składnia polecenia **elif** wygląda identycznie jak w przypadku **if**.

Jeśli nasz program ma sprawdzać kilka warunków (kilka przypadków) najpierw zawsze używamy polecenia if do sprawdzenia pierwszego warunku, zaś przy kolejnych warunkach, korzystamy z polecenia elif.

Polecenie **else** nie wymaga podania żadnych warunków w nawiasie i występuje ono zawsze w parze z poleceniem if. Dzieje się tak dlatego, że polecenia, które będą miały się wykonać będą wykonywane zawsze jeśli nie będą spełnione poprzednie warunki (z if lub z elif).

Przejdźmy do przykładu. Załóżmy, że nasz program po wpisaniu przez użytkownika cyfry 1 w wyskakującym okienku, narysuje kwadrat. Jeśli zaś użytkownik wciśnie 2, zostanie narysowany okrąg.

```
1:  import turtle
2:
3:  jakaFigura = turtle.numinput("Co narysować?", "1 - kwadrat, 2
    - koło")
```

```
4:
5:  if jakaFigura == 1:
6:      for x in range(4):
7:          turtle.fd(50)
8:          turtle.lt(90)
9:
10: elif jakaFigura == 2:
11:     turtle.circle(50)
12:
13: turtle.exitonclick()
```

Listing 10.1

W linijce 3 zmiennej **jakaFigura** przypisujemy wartość jaką wpisze użytkownik w wyskakującym okienku. Następnie w linijce 5 dodajemy instrukcję warunkową **if** i warunek, że jeżeli zmienna **jakaFigura** (to co wpisuje użytkownik) wynosi 1, to zostanie narysowany kwadrat o boku 50 (rysowanie przy pomocy pętli for).

Ponieważ mamy do sprawdzenia jeszcze jeden warunek (czy użytkownik wpisał 2), to w linijce 10 dodaliśmy jeszcze polecenie **elif** i kod jaki ma się wykonać (linijka 11 – rysowanie okręgu o promieniu 50) jeśli warunek taki będzie prawdziwy (czyli jeśli użytkownik wpisze cyfrę 2).

UWAGA: Przy sprawdzaniu warunków korzystamy z dwóch znaków równości ==, jest to tak zwany symbol **porównania** a nie przypisania, gdzie stosowany jest pojedynczy znak równości (zobacz linia 3 gdzie zmiennej przypisujemy określoną wartość wprowadzoną przez użytkownika).

Gdybyśmy chcieli nasz program uzupełnić o rysowanie trójkąta, jeśli użytkownik wpisze wartość liczbową inną niż 1 i 2, to należałoby użyć jeszcze polecenia **else**, a kod programu wyglądałby następująco:

```
1:  import turtle
2:
3:  jakaFigura = turtle.numinput("Co narysować?", "1 – kwadrat, 2
    – koło")
4:
5:  if jakaFigura == 1:
6:      for x in range(4):
7:          turtle.fd(50)
8:          turtle.lt(90)
9:
10: elif jakaFigura == 2:
```

```
11:     turtle.circle(50)
12:
13: else:
14:     turtle.fd(50)
15:     turtle.lt(120)
16:     turtle.fd(50)
17:     turtle.lt(120)
18:     turtle.fd(50)
19:     turtle.lt(120)
20:
21: turtle.exitonclick()
```

Listing 10.2

W rozdziale **ZADANIA** znajdziesz **Zadanie 10**, które wykonane samodzielnie pozwoli lepiej zrozumieć poznane w tym rozdziale zagadnienie.

11. Sterowanie żółwiem przy pomocy klawiatury

Do tej pory nasz żółw poruszał się po wirtualnej tablicy zgodnie z tym jak wcześniej zaplanowaliśmy (zaprogramowaliśmy). Jedynym wyjątkiem było wprowadzanie danych przez użytkownika, ale nawet wówczas wiedzieliśmy, co żółw będzie rysować.

A co jeśli chcielibyśmy sterować naszym żółwiem przy pomocy klawiatury (jak w grze) i po wciśnięciu określonego klawisza żółw chodziłby do przodu, skręcał itd. Jest to oczywiście możliwe i jedynie będziemy musieli skorzystać z funkcji:

turtle.onkeypress()

gdzie w nawiasie będziemy musieli wpisać dwa parametry oddzielone przecinkiem. Pierwszy parametr to nazwa funkcji, którą będziemy chcieli uruchomić, a drugi pisany w cudzysłowie z nazwą klawisza, który będzie tą funkcję uruchamiał. Z języka angielskiego ***on key press*** w dosłownym tłumaczeniu znaczy po prostu ***po naciśnięciu klawisza***.

Na koniec pamiętać jeszcze tylko musimy o uruchomieniu funkcji „nasłuchującej" czy jakiś klawisz został wciśnięty. Funkcja, która tym się zajmuje to:

turtle.listen()

Z języka angielskiego ***listen*** znaczy ***słuchać***. W funkcji tej w nawiasie nie wpisujemy żadnych parametrów.

Poniższy przykład, w którym klawisz z literką „**w**” będzie przesuwał żółwia o 10 kroków, pomoże lepiej zrozumieć to zagadnienie.

```
1: import turtle
2:
3: def doPrzodu():
4:         turtle.fd(10)
5:
6: turtle.onkey(doPrzodu, "w")
7: turtle.listen()
8:
9: turtle.exitonclick()
```

Listing 11

W rozdziale **ZADANIA** znajdziesz **Zadanie 11**, które wykonane samodzielnie pozwoli lepiej zrozumieć poznane w tym rozdziale zagadnienie.

12. Wprowadzenie do liczb losowych

W życiu codziennym spotkaliście się zapewne nie jeden raz z liczbami losowymi, które otrzymujemy w wyniku losowania, czyli na przykład rzutu kostką do gry. W programowaniu z takimi liczbami będziecie spotykać się równie często, a być może nawet częściej. Liczby losowe można uzyskiwać (generować) za pomocą programów komputerowych, „powstają” one w wyniku działania złożonych algorytmów i dlatego bardzo często nazywa się je liczbami pseudolosowymi. Nazywa się je tak dlatego, że posiadają one cechy liczb prawdziwie losowych ale charakteryzują się również pewną ukrytą regularnością.

W niniejszym rozdziale poznamy jeden ze sposobów generowania liczb losowych w Pythonie. Posłużymy się tutaj przykładem programu, który losuje trzy razy wartości liczbowe od 0 do 255, a następnie wylosowane cyfry przypisuje odpowiedniemu nasyceniu barw w palecie kolorów RGB. Znaczy to tyle, że nasz żółw będzie korzystał z losowego koloru podczas rysowania.

Na początek, aby móc korzystać z liczb losowych musimy do naszego programu zaimportować odpowiednią bibliotekę. Służy do tego polecenie:

```
import random
```

Z języka angielskiego ***random*** znaczy po prostu ***losowy***.

Po zaimportowaniu biblioteki random możemy przejść do losowania liczby. Posłużymy się tutaj poleceniem:

random.randint(0, 255)

Randint oznacza, że losowana będzie liczba całkowita (bez przecinków), wartości w nawiasie wskazują zaś przedział (zakres) z jakiego będzie losowana liczba. W naszym przypadku będzie to zatem liczba od 0 do 255.

Zajrzyjmy do przykładu:

```
1:  import turtle
2:  import random
3:
4:  turtle.colormode(255)
5:
6:  wylosowanaLiczba = random.randint(0, 255)
7:  turtle.pencolor(wylosowanaLiczba,0,0)
8:
9:  for x in range(4):
10:     turtle.fd(100)
11:     turtle.lt(90)
12:
13: turtle.exitonclick()
```

Listing 12

W powyższym przykładzie zaimportowaliśmy odpowiednie biblioteki (linia 1 i 2), ustawiliśmy tryb koloru (linia 4) i zmiennej o nazwie **wylosowanaLiczba** przypisaliśmy wylosowaną wartość z zakresu od 0 do 255 (linia 6). Następnie ustawiliśmy kolor pióra (linia 7) na taki, gdzie w palecie RGB, wartości *green* i *blue* ustawiliśmy na 0, zaś wartość *red* na wartość zmiennej **wylosowanaLiczba**, Zatem jeśli tylko wartość *red* będzie wartością losową to pióro naszego żółwia będzie w kolorze od czarnego (jeśli wylosowana wartość to 0) do czerwonego (gdy wylosowana wartość to 255). Na koniec w pętli for zaprogramowaliśmy rysowanie kwadratu o boku 100 (linia 9-11).

Uruchom program z Listingu 12 kilka razy aby zaobserwować jak zmienia się kolor pióra za każdym razem kiedy na nowo uruchamiamy program.

W rozdziale **ZADANIA** znajdziesz **Zadanie 12**, które pozwoli na praktyczne wykorzystanie wiedzy z tego rozdziału.

13. Wyświetlanie tekstu i liczb

Na końcu poprzedniego rozdziału, aby sprawdzić, że losowanie koloru naprawdę działa, trzeba było kilka razy uruchomić program. Prostszym sposobem byłoby jednak wyświetlenie wylosowanej wartości na przykład w formie tekstu w okienku w którym nasz żółw rysuje i jedynie dwukrotnie uruchomienie programu, aby zobaczyć, że wylosowane wartości są zupełnie różne (oczywiście może zdarzyć się, jednak rzadko, że wylosowane liczby dwa razy pod rząd będą identyczne).

Do wyświetlania tekstu czy liczb, jak również i wartości zmiennych, posłużymy się metodą:

turtle.write()

W metodzie tej, w nawiasie wpisujemy odpowiednie parametry oddzielając je przecinkiem.

Pierwszy, w zasadzie jedyny obowiązkowy parametr, to tekst, który chcemy wyświetlić. Piszemy go w cudzysłowie (jeśli jest to tekst przez nas z góry ustalony, który nie będzie się zmieniał w trakcie działania programu) lub w przypadku zmiennej – wpisujemy nazwę zmiennej bez cudzysłowu. Zerknijmy do przykładów:

```
1:  import turtle
2:
3:  turtle.write("Hello")
4:
5:  turtle.exitonclick()
```

Listing 13

```
1:  import turtle
2:  import random
3:
4:  losowaLiczba = random.randint(1,100)
5:  turtle.write(losowaLiczba)
6:
7:  turtle.exitonclick()
```

Listing 14

W pierwszym przykładzie wyświetliliśmy tylko napis „**Hello**”, z kolei w drugim - wartość zmiennej **losowaLiczba**.

A co jeśli chcielibyśmy wyświetlić tekst „**Wylosowana liczba to**:” i wylosowaną liczbę. Jest to możliwe, musimy jednak wówczas dokonać tak zwanej konwersji typu zmiennej (patrz podpowiedź do zadania 9). Musimy zatem wartość liczbową ze zmiennej **losowaLiczba** zamienić na tekst korzystając z polecenia **str(losowaLiczba)**. Teksty i wartości zmiennych łączymy za pomocą znaku +. Poniżej rozwiązanie:

```
1:  import turtle
2:  import random
3:
4:  losowaLiczba = random.randint(1,100)
5:  turtle.write("Wylosowana liczba to: " + str(losowaLiczba))
6:
7:  turtle.exitonclick()
```

Listing 15

Po uruchomieniu programu z listingu 15 zapewne będziemy chcieli jeszcze skorzystać z pozostałych parametrów i w metodzie **write** określić jeszcze rodzaj czcionki, jej wielkość oraz czy ma to być tekst normalny (normal), pogrubiony (bold) czy pochylony (italic). Gdybyśmy chcieli w powyższym przykładzie zastosować pogrubioną czcionkę Arial o wielkości 20, to linijka 5 z listingu 15 wyglądałaby następująco:

```
5:  turtle.write("Wylosowana liczba to: " + str(losowaLiczba),
            font=('Arial', 20, 'bold'))
```

Możemy również zdecydować o wyrównaniu tekstu względem żółwia i tak gdybyśmy chcieli aby tekst wyświetlał się na lewo od niego użylibyśmy parametru **align='left'**, na prawo - **align='right'**, a gdyby żółw miał być dokładnie na środku napisu to: **align='center'**. Wycentrowanie tekstu względem żółwia w listingu 15 wyglądałoby w ten sposób:

```
5:  turtle.write("Wylosowana liczba to: " + str(losowaLiczba),
            font=('Arial', 20, 'bold'), align='center')
```

W rozdziale **ZADANIA** znajdziesz **Zadanie 13**, które pozwoli na przetestowanie swojej wiedzy w praktyce.

ZADANIA

Poniżej znajdziesz kilka prostych zadań, które pozwolą na utrwalenie wiadomości i sprawdzenie się. Postaraj się rozwiązać je samodzielnie. W razie trudności, przeanalizuj jeszcze raz treść poszczególnych rozdziałów, gdyby jednak nadal był problem z ich rozwiązaniem, na kolejnych stronach znajdziesz listingi z przykładowymi rozwiązaniami. Przykładowymi, ponieważ w szczegółach nasze programy mogą się różnić, ważne żeby efekt końcowy, efekt działania, się zgadzał.

Zadanie 1

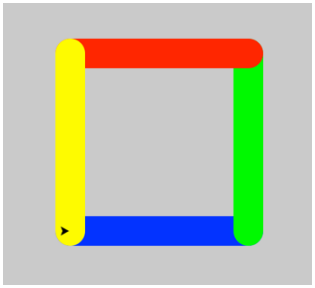
Napisz program, którego zadaniem będzie rysowanie trójkąta równobocznego o długości boku 150.

Zadanie 2

Napisz program, w którym użyjesz dwóch zmiennych: bok_a i bok_b. Przypisz im odpowiednio wartości 100 i 50, a następnie użyj ich w kodzie odpowiedzialnym na rysowanie prostokąta o bokach 100 i 50.

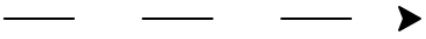
Zadanie 3.1

Stwórz program, który rysuje kwadrat o długości boku 150. Każdy bok kwadratu ma być narysowany innym kolorem oraz linią o grubości 25 pikseli. Zmień również kolor tła na kolor szary.



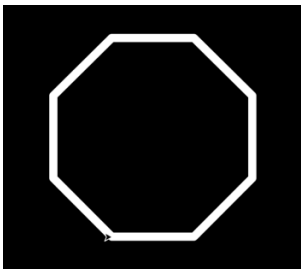
Zadanie 3.2

Napisz program, którego zadaniem będzie narysowanie linii przerywanej, tak jak na poniższym obrazku. Linia ciągła o długości 30 pikseli, a następnie przerwa, również o długości 30 pikseli.



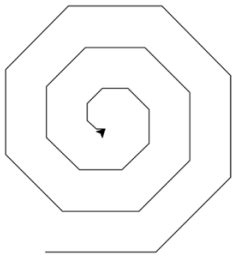
Zadanie 4

Napisz program z wykorzystaniem pętli for, który narysuje ośmiokąt foremny o boku 100. Kolor pióra ustaw na biały, a jego szerokość na 10. Kolor tła ustaw na czarny. Efekt działania programu przedstawiono na poniższym zdjęciu.



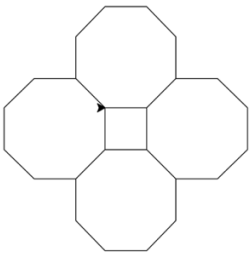
Zadanie 5

Napisz program, którego zadaniem będzie narysowanie spirali – labiryntu, w którym długość pierwszej linii wynosić będzie 130, zaś kąt skrętu 45. Spirala labirynt składać ma się z 25 linii. Długość każdej kolejnej linii ma być mniejsza od poprzedniej o 5 pikseli. W programie zastosuj pętlę for oraz zmienną **dlugosc_linii**.



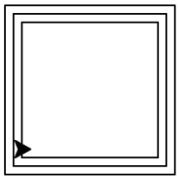
Zadanie 6.1

Napisz program, który narysuje 4 ośmiokąty foremne o boku 50 pikseli, tak jak na poniższym obrazku. W programie zastosuj dwie funkcje. Jedną (np. **def osmiokat()**), która będzie odpowiadać za rysowanie ośmiokąta oraz drugą (np. **def przejście()**), która będzie odpowiadać, za umieszczanie żółwia w odpowiednim miejscu by móc rysować kolejny ośmiokąt. Czterokrotne wywołanie funkcji umieść w pętli for.



Zadanie 6.2

Stwórz program, który będzie rysował, tak jak na zdjęciu poniżej, kwadraty w kwadracie. W sumie mają zostać narysowane trzy kwadraty. Zacznij rysowanie od największego kwadratu, którego bok wynosi 100. Długość boku drugiego kwadratu wynosi 90, trzeciego 80. Czyli długość boku, każdego kolejnego kwadratu zmniejszamy o 10. Odstępy między kwadratami wynosić mają 5 pikseli. Zastosuj pętle for oraz funkcje. Pamiętaj, aby kwadraty się nie dotykały – skorzystaj z poleceń penup() i pendown().

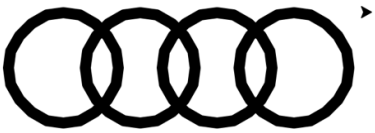


Zadanie 7

Stwórz program, którego zadaniem będzie narysowanie flagi Polski. Flaga powinna zachować proporcje, zatem dłuższy bok powinien mieć długość 160, zaś krótszy 100. Zauważ, że flaga Polski zbudowana jest z dwóch prostokątów o bokach 50x160. „Górny" prostokąt wypełniony jest kolorem białym, zaś „dolny" kolorem czerwonym. W rozwiązaniu tego zadania spróbuj wykorzystać funkcje.

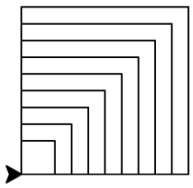
Zadanie 8

Napisz program, który narysuje logo firmy motoryzacyjnej Audi składające się z czterech nachodzących na siebie okręgów. Szerokość pióra ustaw na 10 pikseli.



Zadanie 9

Napisz program rysujący kwadraty w kwadracie (ze wspólnym lewym dolnym rogiem). Długość boku największego kwadratu wynosi 100, natomiast każdego kolejnego jest mniejsza o 10. Dodaj funkcję numinput() aby podczas uruchomienia programu, użytkownik określił, ile kwadratów ma zostać narysowanych. Wykorzystaj parametry opcjonalne i określ wartość minimalną na 2, zaś maksymalną na 9.



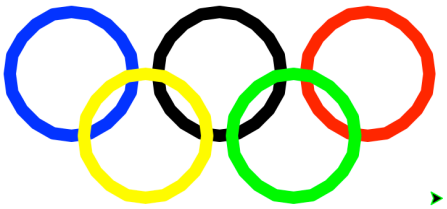
Uwaga podpowiedź: Ponieważ w pętli for w nawiasie może być tylko liczba całkowita, (nie możemy powtarzać określonych poleceń na przykład tylko 3,5 razy) przypisanie zmiennej wartości z funkcji numinput() trzeba nieco zmodyfikować, bowiem funkcja ta pobiera liczbę w postaci liczby z przecinkiem, w momencie, kiedy będziemy chcieli wartość tej zmiennej użyć w pętli for w nawiasie, zostaniemy poinformowani o błędzie w kodzie i program się nie uruchomi.
Rozwiązaniem tego problemu jest tak zwana konwersja typu zmiennej. Dlatego przy pobieraniu danych od użytkownika użyj następującej linii kodu:

```
ileKwadratow = int(turtle.numinput("Ile kwadratów","Podaj ile kwadratów narysować:",
default=2, minval=2, maxval=9))
```

Int przed nawiasem oznacza po prostu, że nawet jeśli użytkownik w wyskakującym okienku poda wartość np. 100, to Python **nie będzie** w zmiennej **ileKwadratow** przechowywać wartości 100.00 tylko zwykłe 100 (bez przecinka).

Zadanie 10

Napisz program, który narysuje koła olimpijskie. Ustaw szerokość pióra na 10. Zadbaj o odpowiednie kolory. Ustaw promień kół na 50. Zadanie wykonaj w ten sposób, że w funkcji wstawisz kod rysowania koła. Samo rysowanie kół wykonasz za pomocą pętli for i w zależności, które z kolei koło będzie rysowane, będzie zmieniał się kolor pióra (np. jeśli rysowane jest koło pierwsze -> użyj koloru niebieskiego). Konieczne tu będzie zastosowanie instrukcji warunkowej z if i elif.

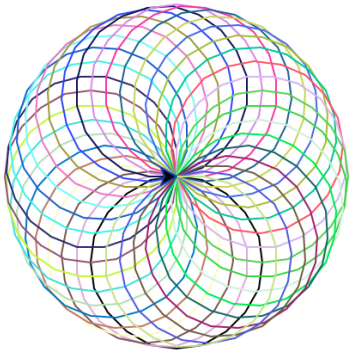


Zadanie 11

Stwórz program, który będzie umożliwiał sterowanie żółwiem przy pomocy klawiatury. Wciśnięcie litery **w** na klawiaturze będzie powodowało przesunięcie żółwia o 10 kroków do przodu, litery **s** – 10 kroków do tyłu, litery **a** – skręt o 10 stopni w lewo, **d**– skręt o 10 stopni w prawo. Klawisz **z** – będzie podnosić pióro, z kolei klawisz **x** – będzie pióro opuszczać.

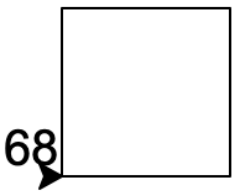
Zadanie 12

Stwórz program, który narysuje kształt złożony z 36 okręgów o promieniu 50. Po narysowaniu okręgu, żółw skręca w lewo o 10 kroków, tak aby kolejny okrąg rysowany był w innym miejscu. Zastosuj do tego pętlę for. W programie wykorzystaj również losowe kolory, tak aby każdy z okręgów narysowany był w kolorze wylosowanym przez program.



Zadanie 13

Napisz program, który wylosuje wartość liczbową z przedziału od 10 do 250 (włącznie) a następnie narysuje kwadrat o takiej długości boku jak wylosowana liczba. Wykorzystaj pętlę for. Na lewo od żółwia wyświetl wylosowaną wartość (czcionką Arial, o wielkości 20).



PRZYKŁADOWE ROZWIĄZANIA ZADAŃ:

Zadanie 1

```
1:  import turtle
2:
3:  turtle.forward(150)
4:  turtle.left(120)
5:  turtle.forward(150)
6:  turtle.left(120)
7:  turtle.forward(150)
8:  turtle.left(120)
9:
10: turtle.exitonclick()
```

Zadanie 2

```
1:  import turtle
2:
3:  bok_a = 100
4:  bok_b = 50
5:
6:  turtle.forward(bok_a)
7:  turtle.left(90)
8:  turtle.forward(bok_b)
9:  turtle.left(90)
10: turtle.forward(bok_a)
11: turtle.left(90)
12: turtle.forward(bok_b)
13: turtle.left(90)
14:
15: turtle.exitonclick()
```

Zadanie 3.1

```
1:  import turtle
2:
3:  turtle.bgcolor("grey")
4:  turtle.pensize(25)
5:
6:  turtle.pencolor("blue")
7:  turtle.forward(150)
8:  turtle.left(90)
9:  turtle.pencolor("green")
10: turtle.forward(150)
11: turtle.left(90)
12: turtle.pencolor("red")
13: turtle.forward(150)
14: turtle.left(90)
15: turtle.pencolor("yellow")
16: turtle.forward(150)
17: turtle.left(90)
18:
19: turtle.exitonclick()
```

Zadanie 3.2

```
1:  import turtle
2:
3:  turtle.forward(30)
4:  turtle.penup()
5:  turtle.forward(30)
6:  turtle.pendown()
7:  turtle.forward(30)
8:  turtle.penup()
9:  turtle.forward(30)
10: turtle.pendown()
11: turtle.forward(30)
12: turtle.penup()
13: turtle.forward(30)
14: turtle.pendown()
15:
16: turtle.exitonclick()
```

Zadanie 4

```
1:  import turtle
2:
3:  turtle.bgcolor("black")
4:  turtle.pencolor("white")
5:  turtle.pensize(10)
6:
7:  for x in range(8):
8:      turtle.forward(100)
9:     turtle.left(45)
10:
11:  turtle.exitonclick()
```

Zadanie 5

```
1:  import turtle
2:
3:  dlugosc_linii = 130
4:
5:  for x in range(25):
6:      turtle.forward(dlugosc_linii)
7:      turtle.left(45)
8:      dlugosc_linii = dlugosc_linii - 5
9:
10: turtle.exitonclick()
```

Zadanie 6.1

```
1:  import turtle
2:
3:  def osmiokat():
4:      for x in range(8):
5:          turtle.forward(50)
6:          turtle.left(45)
7:
8:  def przejscie():
9:      turtle.forward(50)
10:     turtle.right(90)
11:
12: for x in range(4):
13:     osmiokat()
14:     przejscie()
15:
16: turtle.exitonclick()
```

Zadanie 6.2

```
1:  import turtle
2:
3:  bok_a = 100
4:
5:  def kwadrat():
6:      for x in range(4):
7:          turtle.fd(bok_a)
8:          turtle.lt(90)
9:
10: def wchodzenieDoSrodka():
11:     turtle.fd(5)
12:     turtle.lt(90)
13:     turtle.fd(5)
14:     turtle.rt(90)
15:
16: for x in range(3):
17:     kwadrat()
18:     turtle.penup()
19:     wchodzenieDoSrodka()
20:     turtle.pendown()
21:
22:     bok_a = bok_a - 10
23:
24: turtle.exitonclick()
```

Zadanie 7

```
1:  import turtle
2:
3:  def prostokat():
4:      for x in range(2):
5:        turtle.fd(160)
6:        turtle.lt(90)
7:        turtle.fd(50)
8:        turtle.lt(90)
9:
10: def obrot():
11:     turtle.rt(90)
12:     turtle.fd(50)
13:     turtle.lt(90)
14:
15: turtle.fillcolor("white")
16: turtle.begin_fill()
17: prostokat()
18: turtle.end_fill()
19: obrot()
20: turtle.fillcolor("red")
21: turtle.begin_fill()
22: prostokat()
23: turtle.end_fill()
24:
25: turtle.exitonclick()
```

Zadanie 8

```
1:  import turtle
2:
3:  turtle.pensize(10)
4:
5:  for x in range(4):
6:      turtle.circle(50)
7:      turtle.pu()
8:      turtle.fd(70)
9:      turtle.pd()
10:
11: turtle.exitonclick()
```

Zadanie 9

```
1:  import turtle
2:
3:  ileKwadratow = int(turtle.numinput("Ile kwadratów?", "Podaj ile
    kwadratów narysować: ", default=2, minval=2, maxval=9))
4:
5:  bok_a = 100
6:
7:  for x in range(ileKwadratow):
8:      for y in range(4):
9:          turtle.fd(bok_a)
10:         turtle.lt(90)
11:     bok_a = bok_a - 10
12:
13: turtle.exitonclick()
```

Zadanie 10

```
1:  import turtle
2:
3:  turtle.pensize(10)
4:  turtle.speed(0)
5:
6:  def rysujKolo():
7:      turtle.circle(50)
8:
9:  def przesunZolwia():
10:     turtle.pu()
11:     turtle.fd(120)
12:     turtle.pd()
13:
14: def przesunZolwiaDoDrugiejLinii():
15:     turtle.pu()
16:     turtle.bk(180)
17:     turtle.rt(90)
18:     turtle.fd(50)
19:     turtle.lt(90)
20:     turtle.pd()
21:
22: for x in range(5):
23:    if (x == 0):
24:        turtle.pencolor("blue")
25:     elif(x == 1):
26:        turtle.pencolor("black")
27:        przesunZolwia()
28:     elif(x == 2):
29:        turtle.pencolor("red")
30:        przesunZolwia()
31:     elif(x == 3):
32:        turtle.pencolor("yellow")
33:        przesunZolwiaDoDrugiejLinii()
34:     elif(x == 4):
35:        turtle.pencolor("green")
36:        przesunZolwia()
37:
38:     rysujKolo()
39:
40: turtle.exitonclick()
```

Zadanie 11

```
1:  import turtle
2:
3:  def doPrzodu():
4:      turtle.fd(10)
5:  def doTylu():
6:      turtle.bk(10)
7:  def wLewo():
8:      turtle.lt(10)
9:  def wPrawo():
10:     turtle.rt(10)
11: def pioroWGore():
12:     turtle.penup()
13: def pioroWDol():
14:     turtle.pendown()
15:
16: turtle.onkeypress(doPrzodu, "w")
17: turtle.onkeypress(doTylu, "s")
18: turtle.onkeypress(wLewo, "a")
19: turtle.onkeypress(wPrawo,"d")
20: turtle.onkeypress(pioroWGore, "z")
21: turtle.onkeypress(pioroWDol, "x")
22: turtle.listen()
23:
24: turtle.exitonclick()
```

Zadanie 12

```
1:  import turtle
2:  import random
3:  turtle.speed(0)
4:  turtle.colormode(255)
5:
6:  for x in range(36):
7:      turtle.circle(50)
8:      turtle.left(10)
9:
10:     liczba1 = random.randint(0, 255)
11:     liczba2 = random.randint(0, 255)
12:     liczba3 = random.randint(0, 255)
13:
14:     turtle.pencolor(liczba1,liczba2,liczba3)
15:
16: turtle.exitonclick()
```

Zadanie 13

```
1:  import turtle
2:  import random
3:
4:  bokA = random.randint(10, 250)
5:
6:  for x in range(4):
7:      turtle.fd(bokA)
8:      turtle.left(90)
9:
10: turtle.write(bokA, font=("Arial", 20, "normal"), align='right')
11:
12: turtle.exitonclick()
```

PYTHON Wprowadzenie do programowania z wykorzystaniem biblioteki turtle, to książka dla osób rozpoczynających swoją przygodę z programowaniem. W przystępny sposób, na przykładzie tworzenia (kodowania rysowania) określonych figur geometrycznych i kształtów, wyjaśnia najważniejsze pojęcia związane z programowaniem, takie jak chociażby: zmienne, pętle, funkcje czy instrukcje warunkowe. Aby ułatwić zrozumienie poszczególnych zagadnień, do każdego rozdziału przypisano zadanie do samodzielnego wykonania, co już od samego początku pozwoli Czytelnikowi poczuć się twórcą i zachęci do dalszego zgłębiania wiedzy na temat najpopularniejszego obecnie języka programowania jakim jest Python. Na końcu książki znajdują się przykładowe rozwiązania zadań, które pozwolą na porównanie ich z własnymi i zweryfikowanie swoich umiejętności i wiedzy.

Mariusz Nierzwicki – na co dzień nauczyciel informatyki ale również i historii. Pasjonat programowania w językach Java i Python. Twórca takich aplikacji mobilnych jak: „Wieki” i „Cyfry rzymskie”. Prowadzi kanał na YouTubie pod nazwą QuickAnswer, gdzie popularyzuje także programowanie w Scratchu. Entuzjasta nowoczesnych technologii oraz szeroko rozumianej robotyki.

ISBN 978-83-964950-0-6
9 788396 495006