Temat 14. Programowanie kompozycji graficznych w języku Python

1. **Tworzenie w języku Python programu wykorzystującego grafikę żółwia**
2. **Stosowanie kolorów i określanie grubości linii**
3. **Stosowanie instrukcji iteracyjnej w grafice żółwia**
4. **Wykorzystanie funkcji do tworzenia kompozycji graficznych**

Warto powtórzyć

1. Na czym polega programowanie?
2. Czym jest język programowania, a czym program komputerowy?
3. Czym jest interpretacja programu?
4. W jaki sposób zapisywaliśmy powtarzające się polecenia w środowiskach programowania Baltie lub Scratch?
5. W jakim celu stosuje się podprogramy? Jak definiowaliśmy podprogramy (procedury) w środowiskach programowania Baltie lub Scratch?

1. Tworzenie w języku Python programu wykorzystującego grafikę żółwia

Aby pisać programy w języku Python, należy zainstalować darmową aplikację Python, w której skład wchodzi zintegrowane środowisko programistyczne **IDLE**. Program jest dostępny na stronie internetowej *https://www.python.org/downloads/*.

IDLE umożliwia pisane programów, zapisywanie ich i uruchamianie. Zawiera m.in.: powłokę Pythona (rys. 3a), edytor kodu źródłowego (rys. 3b), interpreter.

Uwaga

W tym temacie korzystamy z aplikacji Python w wersji 3.11.1.

Rys. 1. Oficjalna strona internetowa, służąca m.in. do pobierania aplikacji Python: *https://www.python.org/downloads/*

Program w języku Python będziemy tworzyć w oknie edytora kodu źródłowego. Język Python, tak jak każdy język programowania, ma swój zbiór instrukcji i **słów kluczowych** oraz posiada odpowiednie zasady składni i właściwe słownictwo. Instrukcje realizują czynności, takie jak: wprowadzanie danych, wyprowadzanie wyników, wykonywanie obliczeń, określanie warunków czy powtarzanie operacji. W tym temacie omówimy wybrane instrukcje i słowa kluczowe przydatne do programowania kompozycji graficznych.

S Słowo kluczowe

Słowo mające szczególne znaczenie w danym języku programowania, np. oznaczające określony rozkaz, instrukcję lub polecenie.

W języku Python są także gotowe moduły z funkcjami ułatwiającymi programowanie. Jednym z nich jest moduł `turtle` (pol. *żółw*), zawierający funkcje umożliwiające programowanie grafiki. Obraz na ekranie powstaje po zaprogramowaniu ruchów żółwia – podobnie jak po zaprogramowaniu ruchów duszka w języku Scratch (rys. 2.).

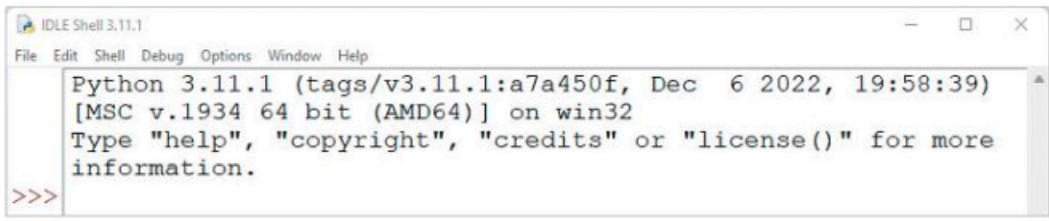
Rys. 2. Polecenia języka Scratch rysujące pokazaną obok figurę

W każdym programie w języku Python wykorzystującym grafikę żółwia przed poleceniami tworzącymi tę grafikę umieszczamy instrukcję **`import`** `turtle`, które określa, że będziemy z takich poleceń korzystać.

Jeżeli chcemy wykorzystać funkcję z modułu `turtle`, należy w programie umieścić nazwę modułu (`turtle`), a po niej nazwę funkcji graficznej – rozdzielone kropką, na przykład `turtle.forward(100)`.

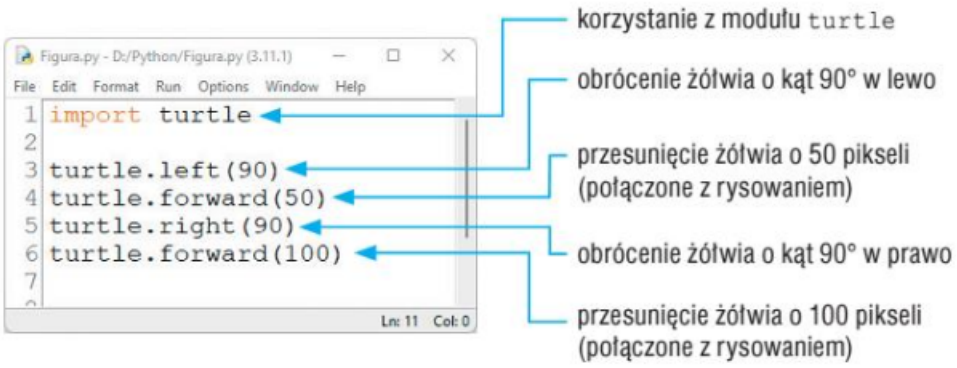
Etapy tworzenia programu w języku Python

1. Uruchamiamy IDLE, wybierając w menu **Start** aplikację **Python 3.11/IDLE (Python 3.11 64-bit)** – otworzy się okno edytora kodu źródłowego.
2. W otwartym oknie powłoki Pythona klikamy opcję **File/New File** (rys. 3a) – otworzy się okno edytora kodu źródłowego.



Rys. 3a. Okno powłoki Pythona

3. W otwartym oknie edytora kodu źródłowego piszemy program (rys. 3b).
4. Zapisujemy program w pliku, wybierając w oknie edytora kodu źródłowego opcję **File/Save as**.

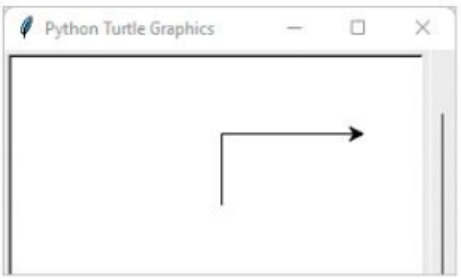


Rys. 3b. Okno edytora kodu źródłowego z przykładowym programem w języku Python

5. Uruchamiamy program, wybierając w oknie edytora kodu źródłowego opcję **Run/Run module** – w przypadku używania modułu `turtle` wynik działania programu pokaże się w oknie **Python Turtle Graphics** (rys. 3c).

Uwaga

Domyślnie żółw jest reprezentowany przez strzałkę (na początku skierowaną w prawą stronę). Aby zamienić ją na ikonę żółwia, należy umieścić w programie (po instrukcji `import turtle`) polecenie: `turtle.shape("turtle")`.



Rys. 3c. Okno **Python Turtle Graphics** z wynikiem działania programu z rysunku 3b

Aby wyświetlić numerację wierszy w oknie edytora kodu źródłowego, należy wybrać opcję **Options/Show Line Numbers**.
Po wybraniu opcji **Options/Configure IDLE** można zmienić m.in. rozmiar czcionki.

Ćwiczenie 1. Tworzymy program w języku Python

1. Uruchom IDLE i utwórz program składający się z poleceń pokazanych na rysunku 3b.
2. Zapisz program w pliku pod nazwą *Figura*.
3. Uruchom program. Porównaj efekt jego działania z rys. 3c.
4. Zmodyfikuj program, dodając jeszcze kilka poleceń obracających żółwia o dowolne kąty i przesuwających na wybrane odległości. Zapisz program w pliku pod tą samą nazwą.

Polecenie	Opis	Przykład
`forward(dystans)`	przesuwa żółwia do przodu o podaną liczbę pikseli	`forward(70)`
`backward(dystans)`	przesuwa żółwia do tyłu o podaną liczbę pikseli	`backward(100)`
`right(kąt)`	obraca żółwia w prawo o kąt podany w stopniach	`right(45)`
`left(kąt)`	obraca żółwia w lewo o kąt podany w stopniach	`left(60)`
`circle(promień)`	rysuje okrąg o promieniu podanym w pikselach i środku znajdującym się o *promień* pikseli na lewo od aktualnej pozycji żółwia	`circle(150)`
`pensize(rozmiar)`	określa grubość linii rysowanej przez żółwia (w pikselach)	`pensize(3)`
`pencolor(kolor)`	określa kolor linii rysowanej przez żółwia	`pencolor("red")`
`bgcolor(kolor)`	określa kolor tła całego ekranu	`bgcolor("blue")`
`begin_fill()`	zaczyna wypełnianie figury	`begin_fill()`
`fillcolor(kolor_wypełnienia)`	określa kolor wypełnienia	`fillcolor("green")`
`end_fill()`	kończy wypełnianie figury	`end_fill()`
`color(kolor_linii, kolor_wypełnienia)`	określa kolor linii i kolor wypełnienia	`color("red", "blue")`
`penup()`	podnosi pisak żółwia (żółw nie będzie zostawiał śladu przy poruszaniu się)	`penup()`
`pendown()`	opuszcza pisak żółwia (żółw będzie zostawiał ślad przy poruszaniu się)	`pendown()`
`goto(0,0)`	przemieszcza żółwia do punktu początkowego (do środka ekranu)	`goto(0,0)`
`mainloop()`	uniemożliwia zamknięcie okna z wynikiem działania programu od razu po zakończeniu jego działania	`mainloop()`

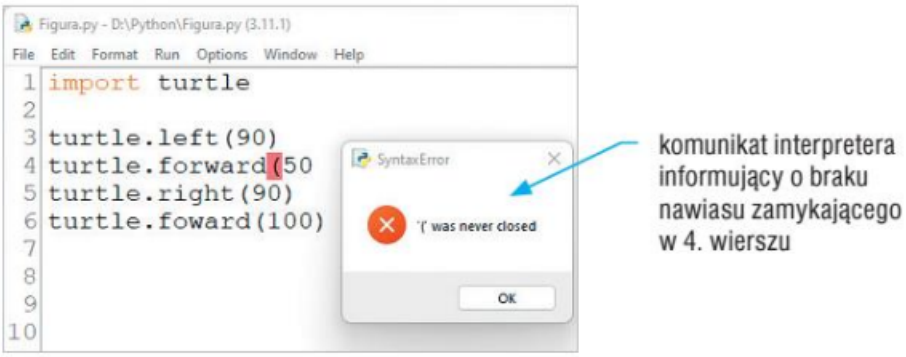
Tabela 1. Wybrane funkcje modułu `turtle`

Musimy pamiętać o poprawnym zapisaniu instrukcji i słów kluczowych w języku Python, a także o odpowiednim wstawieniu nawiasów. W trakcie wykonywania programu interpreter rozpoznaje polecenia języka Python i zamienia je na instrukcje wewnętrznego języka procesora.

Interpreter próbuje wykonać program instrukcja po instrukcji. Jeżeli w programie wystąpi błąd, interpreter przerwie wykonanie programu i wyświetli komunikat o błędzie. Komunikaty mogą pojawiać się w osobnym oknie (rys. 4a) lub w oknie powłoki Pythona (rys. 4b). W programie na rysunku 4a występują dwa błędy (w wierszu 4. i 6.).

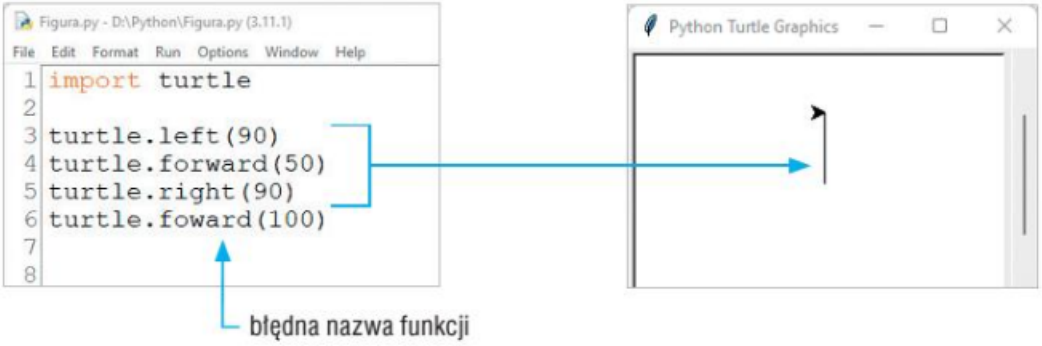
Uwaga

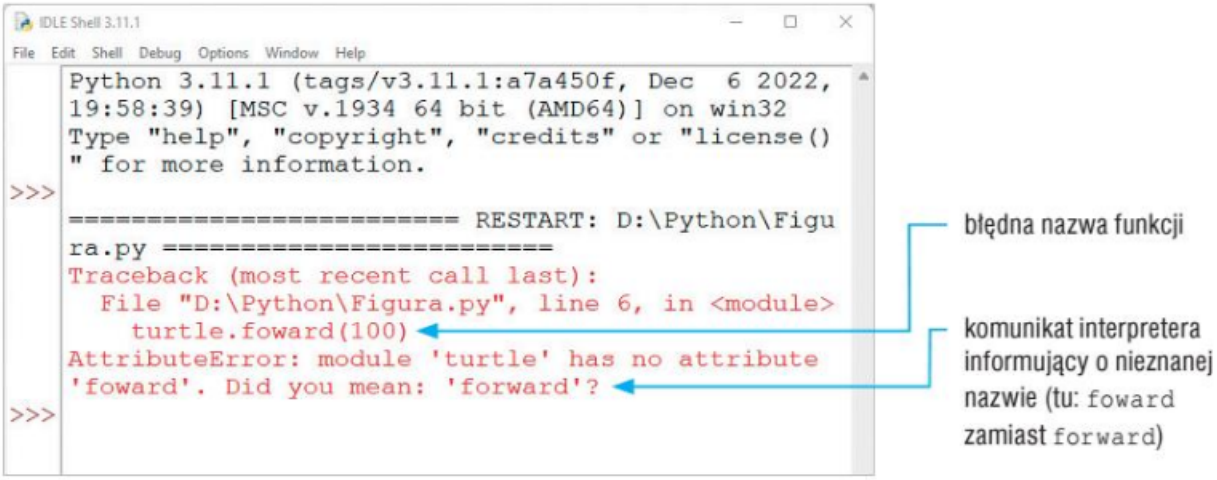
Pusty wiersz wstawiamy w programie dla zwiększenia przejrzystości programu (tu: wiersz 2.).



Rys. 4a. Komunikat interpretera informujący o błędzie

Po poprawieniu błędu w 4. wierszu, ponownym zapisaniu i uruchomieniu programu, zostaną kolejno wykonane poprawne instrukcje (wiersze 3-5), czego efekt zobaczymy na ekranie. Interpreter przerwie program na wierszu 6. i wskaże błąd (rys. 4b).





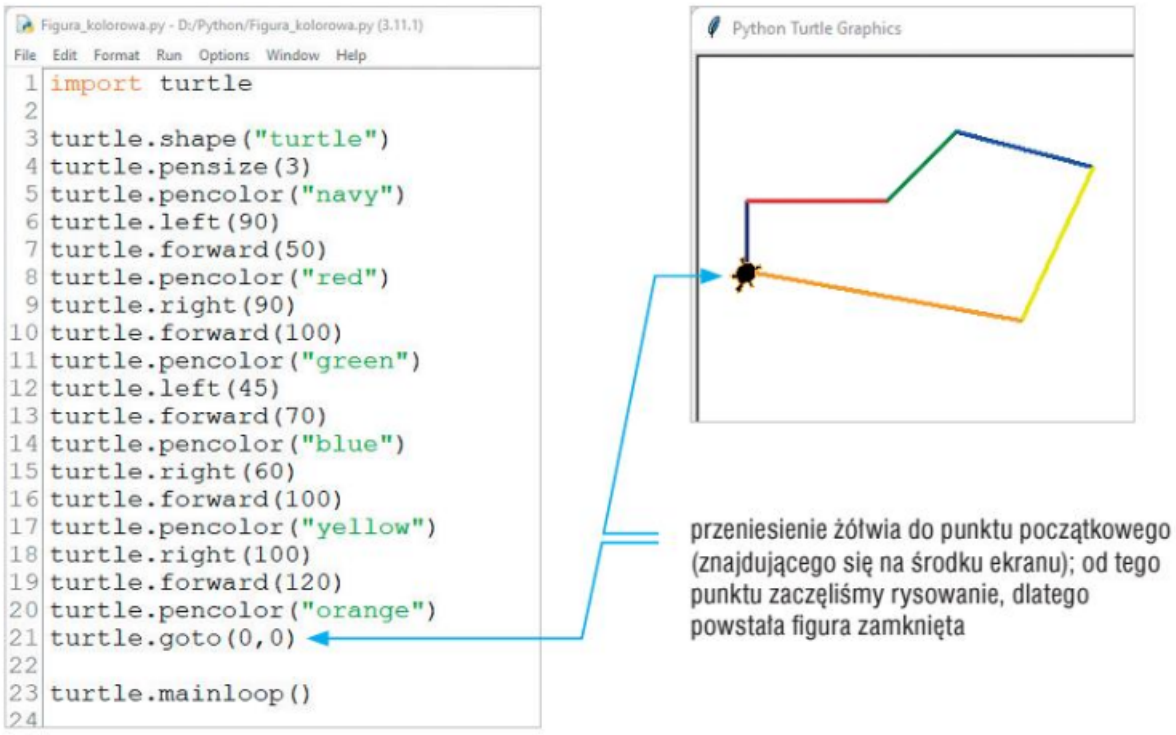
Rys. 4b. Okno powłoki Pythona z komunikatem interpretera informującym o błędzie

Aby unikać popełniania błędów podczas pisania programu, można stosować w edytorze kodu źródłowego operacje kopiowania, wycinania i wklejania fragmentu tekstu z wykorzystaniem **Schowka**, tak jak w każdym innym edytorze tekstu.

2. Stosowanie kolorów i określanie grubości linii

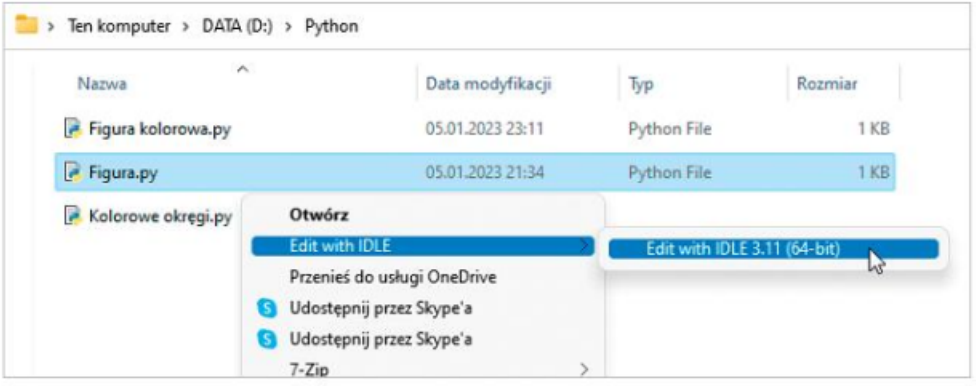
Chcemy, aby żółw rysował linie kolorowymi pisakami o wybranym rozmiarze. Jak zmodyfikować program, aby żółw zostawiał kolorowy ślad wybranej grubości?

Aby zmienić kolor rysowanej linii, należy umieścić w odpowiednim miejscu programu funkcję, która umożliwia zmianę koloru pisaka – `pencolor()`. Wewnątrz nawiasów podajemy nazwę koloru (w języku angielskim) w górnych cudzysłowach. W programie na rysunku 5. wszystkie linie będą miały grubość 3 pikseli (rozmiar pisaka określono w 4. wierszu).



Rys. 5. Przykładowy program z zastosowaniem zmiany kolorów i określania grubości linii oraz efekt jego wykonania

Jeśli zamknęliśmy okno powłoki Pythona, a chcemy edytować program, możemy w oknie Eksploratora plików z menu kontekstowego nazwy pliku wybrać polecenie **Edit with IDLE/ Edit with IDLE 3.11 (64-bit)** – rys. 6.



Rys. 6. Otwieranie okna edytora kodu źródłowego z programem zapisanym w pliku

Po zamknięciu okna powłoki Pythona możemy również uruchomić program, klikając jego nazwę w Eksploratorze plików. Aby okno z wynikiem działania programu (tu: rysunkiem) nie zamknęło się od razu po zakończeniu jego działania, należy umieścić na końcu programu polecenie: `turtle.mainloop()`.

Ćwiczenie 2. Stosujemy kolory i ustalamy grubość linii

1. Otwórz plik *Figura* zapisany w ćwiczeniu 1. Zmodyfikuj program, aby każda linia była rysowana innym kolorem. Program na rysunku 5. może stanowić inspirację.
2. Zapisz program w pliku pod tą samą nazwą i uruchom go.
3. Zamknij okno Pythona i uruchom ponownie program, klikając jego nazwę w oknie Eksploratora plików.

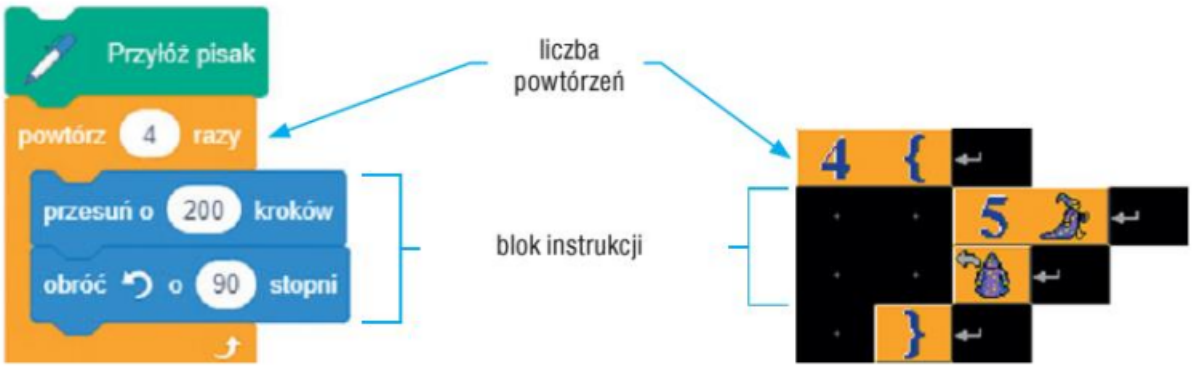
Wskazówka: Aby ułatwić sobie pisanie programu, pamiętaj o możliwości kopiowania poleceń.

3. Stosowanie instrukcji iteracyjnej w grafice żółwia

Chcemy utworzyć w języku Python program rysujący kwadrat.
Jak to zrobić w grafice żółwia?

Podczas rysowania kwadratu powtarzamy cztery razy dwie operacje: rysowania boku kwadratu (czyli przesuwania żółwia o długość boku) i obracania żółwia o kąt 90°. Możemy te polecenia zapisać kolejno tak jak w programach na rysunku 8a, ale nie jest to optymalny sposób programowania. Lepiej zastosować instrukcję, która umożliwi powtarzanie poleceń.

W języku Scratch powtarzające się polecenia zapisywaliśmy, stosując polecenie **powtórz**. Liczbę powtórzeń określaliśmy, zmieniając liczbę w odpowiednim polu tekstowym (rys. 7a). W Baltiem powtarzające się polecenia zapisywaliśmy w nawiasach klamrowych, a przed nawiasem otwierającym umieszczaliśmy liczbę powtórzeń (rys. 7b).



Rys. 7a. Program rysujący kwadrat (Scratch)

Rys. 7b. Program przemieszczający czarodzieja wzdłuż boków kwadratu (Baltie)

W języku Python do zapisywania powtarzających się poleceń możemy wykorzystać instrukcję iteracyjną (zwaną też instrukcją pętli) **`for`**. Do określenia liczby powtórzeń możemy zastosować funkcję `range()`, którą umieszczamy po słowie kluczowym **`in`** (rys. 8b).

```
*Kwadrat_bez_petli.py - D:\Python\Kwadrat_bez_pet...
File  Edit  Format  Run  Options  Window  Help
1  import turtle
2
3  turtle.forward(200)
4  turtle.left(90)
5  turtle.forward(200)
6  turtle.left(90)
7  turtle.forward(200)
8  turtle.left(90)
9  turtle.forward(200)
10
11 turtle.mainloop()
12
```

Rys. 8a. Program w języku Python rysujący kwadrat bez użycia instrukcji iteracyjnej

```
Kwadrat_petla.py - D:\Python\Kwadrat_petla.py (3.11.1)
File  Edit  Format  Run  Options  Window  Help
1  import turtle
2
3  for i in range(4):
4      turtle.forward(200)
5      turtle.left(90)
6
7  turtle.mainloop()
8
```

liczba powtórzeń

blok instrukcji

Rys. 8b. Program w języku Python rysujący kwadrat z użyciem instrukcji iteracyjnej

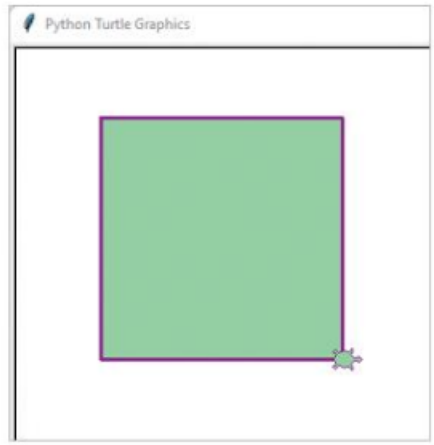
W programie na rysunku 8b funkcja `range(4)` generuje wartości z zakresu od 0 do 3, czyli instrukcje z bloku instrukcji zostaną wykonane cztery razy, a zmienna *i* będzie przyjmować kolejno wartości: 0, 1, 2, 3 (w programie na rysunku 8b z wartości tych jednak nie korzystamy).

W bloku instrukcji możemy umieścić jedną instrukcję lub więcej. Ważne jest, aby instrukcje w bloku były przesunięte przynajmniej o jedną spację w prawo (przyjęte jest stosowanie czterech spacji).

```
Kwadrat.py - D:/Python/Kwadrat.py (3.11.1)
File  Edit  Format  Run  Options  Window  Help
import turtle

turtle.shape("turtle")
turtle.pensize(3)
turtle.color("purple", "light green")
turtle.begin_fill()
for i in range(4):
    turtle.left(90)
    turtle.forward(200)
turtle.end_fill()

turtle.mainloop()
```



Rys. 9. Przykładowy program w języku Python rysujący kolorowy kwadrat i efekt jego wykonania – ćwiczenie 3.

Ćwiczenie 3. Stosujemy instrukcję iteracyjną do narysowania kwadratu

1. Przeanalizuj program pokazany na rysunku 9. Zwróć uwagę na miejsce umieszczenia w programie poleceń dotyczących kolorowania – podaj, jakie jest ich działanie. Zastanów się, dlaczego te polecenia nie muszą być umieszczone w bloku instrukcji wewnątrz instrukcji `for`.
2. Utwórz program rysujący dowolnym kolorem pisaka o grubości 3 pikseli kwadrat wypełniony wybranym kolorem.
3. Zapisz program w pliku pod nazwą *Kwadrat*. Uruchom program.

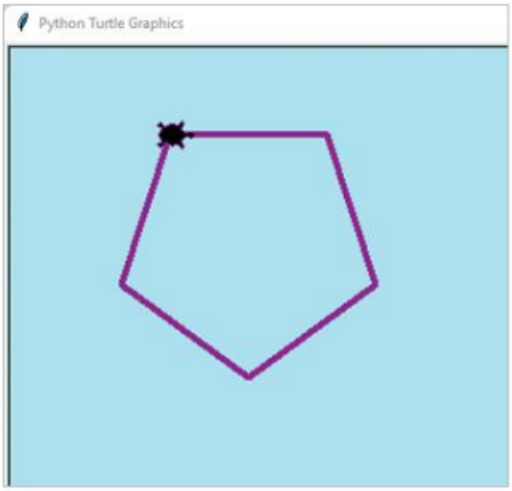
Ćwiczenie 4. Stosowanie instrukcji iteracyjnej do narysowania pięciokąta foremnego

1. Utwórz program rysujący pięciokąt foremny o boku 100. Ustal rozmiar pisaka na 4 piksele, kolor – fioletowy (ang. *purple*). Dodaj jasnoniebieskie tło (ang. *light blue*).
2. Odpowiedz na pytania:
   a. Jakie polecenia powinny być wykonywane w pętli?
   b. Jaki przyjąć kąt obrotu żółwia?
3. Zapisz program w pliku pod nazwą *Pieciokat*. Uruchom program.

Wskazówka: Zobacz podobny program w języku Scratch (temat 12., ćwiczenie 3.).

Uwaga

W języku Python w nazwach plików, zmiennych i funkcji nie będziemy stosować polskich liter.



Rys. 10. Przykładowy efekt wykonania programu – ćwiczenie 4.

4. Wykorzystanie funkcji do tworzenia kompozycji graficznych

Chcemy utworzyć w języku Python program rysujący kompozycję złożoną z wielu figur, np. kwadratów. Jak to zrobić w grafice żółwia?

Aby rysować złożone kompozycje, można wykorzystać instrukcję pętli `for` i funkcje – wbudowane oraz tworzone samodzielnie.

Jedną z wbudowanych funkcji jest funkcja `circle()` należąca do modułu `turtle`, rysująca okrąg. Funkcja posiada jeden **parametr** określający promień okręgu. Początek i koniec okręgu są w miejscu aktualnego położenia żółwia, natomiast środek okręgu jest w odległości długości promienia na lewo od aktualnej pozycji żółwia.

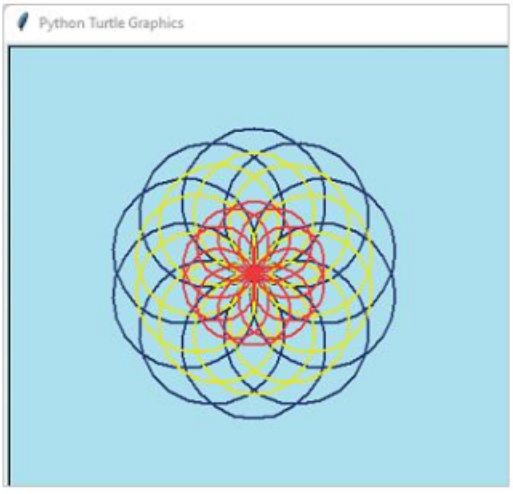
Korzystając z funkcji `circle()`, narysujemy kompozycję składającą się z okręgów o różnych promieniach. W programie na rysunku 11a funkcja `circle()` występuje trzykrotnie z różnymi wartościami promienia.

Kolorowe_okregi.py - D:/Python/Kolorowe_okregi.py (3.11.1)

File Edit Format Run Options Window Help

```
1  import turtle
2
3  turtle.pensize(2)
4  for i in range(10):
5      turtle.color("navy")
6      turtle.circle(60)
7      turtle.color("yellow")
8      turtle.circle(50)
9      turtle.color("red")
10     turtle.circle(30)
11     turtle.right(36)
12 turtle.bgcolor("light blue")
13
14 turtle.mainloop()
15
```

Rys. 11a. Przykładowy program w języku Python rysujący kompozycję z różnokolorowych okręgów – ćwiczenie 5.



Rys. 11b. Efekt wykonania programu z rysunku 11a – ćwiczenie 5.

Ćwiczenie 5. Korzystamy z funkcji modułu `turtle` rysującej okrąg

1. Przepisz program pokazany na rysunku 11a. Zapisz program w pliku pod nazwą *Kolorowe_okregi*.
2. Uruchom program. Objaśnij znaczenie poszczególnych wierszy programu. Które instrukcje są wykonywane w pętli?
3. Zmodyfikuj program, aby najpierw żółw narysował wszystkie granatowe okręgi, potem wszystkie żółte, a na końcu czerwone. Zapisz program w pliku pod nazwą *Kolorowe_okregi2*. Uruchom program.

W języku Python nie ma gotowej funkcji rysującej kwadrat. Aby utworzyć kompozycję z kwadratów, podobną do utworzonej z okręgów (rys. 11b), musielibyśmy w pętli `for` (zaczynającej się w wierszu 4.) umieścić trzy razy wszystkie polecenia rysujące kwadrat.

Z tematu 10. wiemy, że w rozwiązywanym problemie można określić problem cząstkowy w postaci **podprogramu** (w języku Python zwany **funkcją**). W naszym przypadku problemem cząstkowym jest rysowanie kwadratu, dlatego zdefiniujemy funkcję rysującą kwadrat, a następnie połączymy ją z programem głównym poprzez jej **wywołanie**.

W języku Python możemy definiować funkcje bez parametrów (rys. 12a) i z parametrami (rys. 13a).

Aby zastosować funkcję w programie, trzeba ją **zdefiniować**, a następnie **wywołać**:

- w przypadku wywołania funkcji bez parametrów umieszczamy w programie głównym nazwę funkcji z pustymi nawiasami,
- w przypadku wywołania funkcji z parametrami umieszczamy w programie głównym nazwę funkcji z wartościami parametrów w nawiasach.

W języku Python funkcję definiujemy zazwyczaj na początku programu, przed jej pierwszym wywołaniem. Wywołanie funkcji powoduje wykonanie instrukcji będących treścią funkcji.

Ćwiczenie 6. Stosujemy w programie funkcję bez parametrów rysującą kwadrat

1. Przepisz program pokazany na rysunku 12b. Zapisz program w pliku pod nazwą *Kwadrat_funkcja*.
2. Uruchom program. Objaśnij znaczenie poszczególnych wierszy programu. Wskaż miejsce wywołania funkcji. Jakie instrukcje są wykonywane w pętli?
3. Sprawdź, jaką otrzymasz kompozycję, gdy zmienisz liczbę powtórzeń na 10, a kąt obrotu na 36. Zmodyfikuj odpowiednio program. Zapisz program w pliku pod nazwą *Kwadrat_funkcja2*. Uruchom program.

Uwaga !

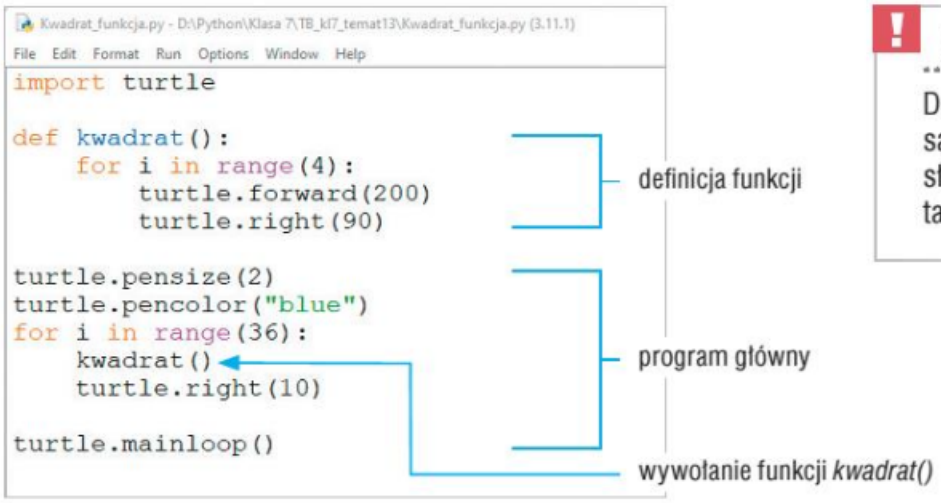
Podprogramy (procedury) tworzyliśmy również w środowiskach Baltie (temat 12.) i Scratch (temat 13.).

```
def kwadrat():
    for i in range(4):
        turtle.forward(200)
        turtle.right(90)
```

nagłówek funkcji

treść funkcji

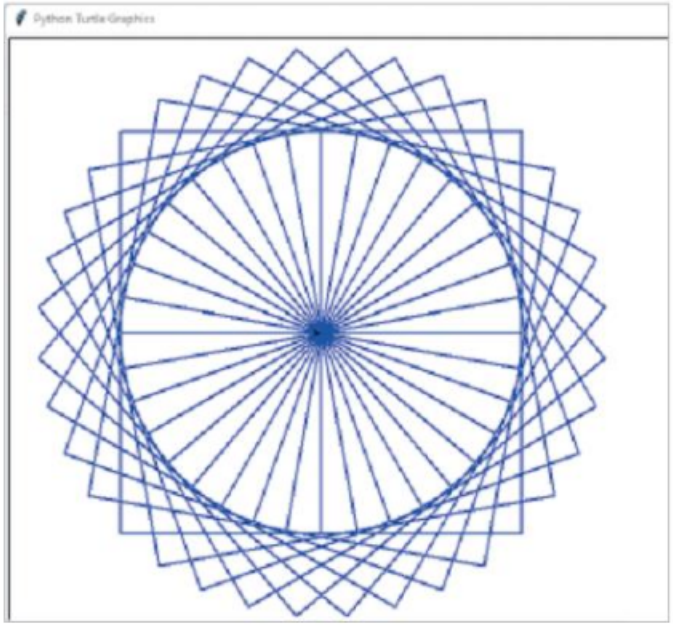
Rys. 12a. Definicja funkcji *kwadrat()* bez parametrów



Uwaga

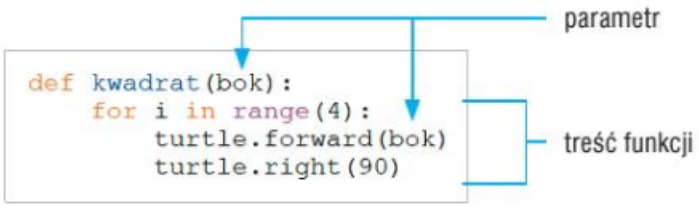
Do nazwy funkcji utworzonej samodzielnie nie dodajemy słowa `turle`, ponieważ funkcja ta nie należy do tego modułu.

Rys. 12b. Program w języku Python z definicją funkcji *kwadrat()* bez parametrów – ćwiczenie 6.



Rys. 12c. Kompozycja z kwadratów o bokach tej samej długości – efekt wykonania programu z rysunku 12b

W ćwiczeniu 6. zdefiniowaliśmy funkcję rysującą kwadrat o boku podanym w definicji (200 pikseli). Aby utworzyć kompozycję podobną do pokazanej na rysunku 13c, musielibyśmy zdefiniować trzy funkcje rysujące kwadraty. W takim przypadku lepiej zdefiniować jedną funkcję z parametrem *bok* (rys. 13a), określającym długość boku kwadratu. Wartość parametru będzie podana w instrukcji wywołania tej funkcji (rys. 13b).



Rys. 13a. Definicja funkcji *kwadrat()* z jednym parametrem

```
Kwadrat_funkcja_parametr.py - D:/Python/Kwadrat_funkcja_parametr.py (3.11.1)
File  Edit  Format  Run  Options  Window  Help
 1 import turtle
 2
 3 def kwadrat(bok):
 4     for i in range(4):
 5         turtle.forward(bok)
 6         turtle.right(90)
 7
 8 turtle.pensize(2)
 9 for i in range(10):
10     turtle.pencolor("dark green")
11     kwadrat(100)
12     turtle.pencolor("orange")
13     kwadrat(50)
14     turtle.pencolor("red")
15     kwadrat(25)
16     turtle.right(36)
17 turtle.bgcolor("light yellow")
18
19 turtle.mainloop()
```

definicja funkcji z jednym parametrem *bok*

wywołanie funkcji z wartością parametru *bok* równą 100

wywołanie funkcji z wartością parametru *bok* równą 50

wywołanie funkcji z wartością parametru *bok* równą 25

Rys. 13b. Program w języku Python z definicją funkcji *kwadrat()* z jednym parametrem – ćwiczenie 7.

Rys. 13c. Kompozycja z kwadratów o różnych długościach boków – efekt wykonania programu z rysunku 13b

Ćwiczenie 7. Stosujemy w programie funkcję z parametrem rysującą kwadrat

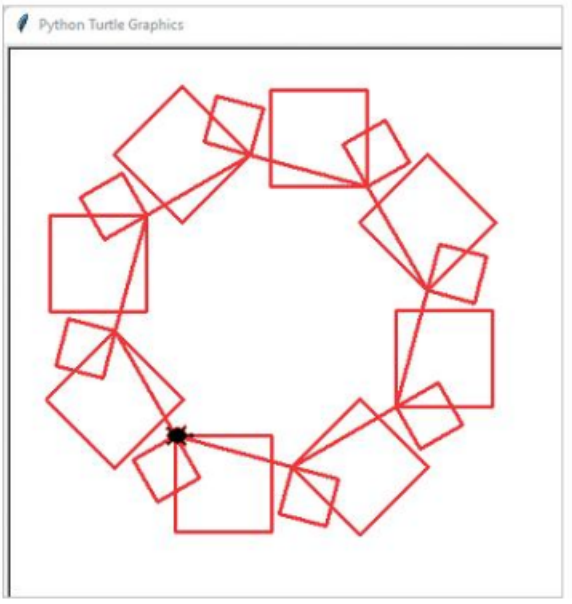
1. Przepisz program pokazany na rysunku 13b. Zapisz program w pliku pod nazwą *Kolorowe_kwadraty*.
2. Uruchom program. Objaśnij znaczenie poszczególnych wierszy programu. Które instrukcje są wykonywane w pętli? Ile razy jest wywołana funkcja *kwadrat()* i z jakimi wartościami?
3. Zmodyfikuj program, aby najpierw żółw narysował wszystkie zielone kwadraty, potem wszystkie żółte, a na końcu czerwone. Zapisz program w pliku pod nazwą *Kolorowe_kwadraty2*. Uruchom program.

Korzystając ze zdefiniowanych funkcji, można tworzyć inne ciekawe kompozycje graficzne (rys. 14.). Możemy również w podobny sposób definiować funkcje rysujące inne wielokąty foremne, np. pięciokąty, i tworzyć z nich kompozycje (rys. 15.).

*Kompozycja_z_kwadratow_c8.py - D:\Python\Klasa 7\TB_kl7_temat13\Kompo

File Edit Format Run Options Window Help

```
1  import turtle
2
3  def kwadrat(bok):
4      for i in range(4):
5          turtle.forward(bok)
6          turtle.right(90)
7
8  turtle.shape("turtle")
9  turtle.pensize(3)
10 turtle.pencolor("red")
11 for i in range(8):
12     kwadrat(80)
13     turtle.right(15)
14     turtle.forward(100)
15     kwadrat(40)
16     turtle.left(60)
17
18 turtle.mainloop()
19
```



Rys. 14. Zastosowanie zdefiniowanej funkcji *kwadrat()* z parametrem do tworzenia kompozycji z kwadratów i efekt wykonania programu – ćwiczenie 8.

Ćwiczenie 8. Stosujemy zdefiniowaną funkcję rysującą kwadrat w innym programie

1. Przepisz program pokazany na rysunku 14. Zapisz program w pliku pod nazwą *Kompozycja_z_kwadratow*.
2. Uruchom program. Objaśnij znaczenie poszczególnych wierszy programu. Ile razy jest wywoływana funkcja *kwadrat(80)*, a ile – *kwadrat(40)*? Który kwadrat (o jakim boku) jest rysowany jako pierwszy, a który jako ostatni?
3. Zmodyfikuj program główny, aby powstała nowa kompozycja – według twojego pomysłu. Zapisz program w pliku pod nazwą *Moja_kompozycja*.

Rys. 15. Kompozycja z pięciokątów foremnych – ćwiczenie 9.

Ćwiczenie 9. Definiujemy funkcję rysującą pięciokąt foremny i tworzymy kompozycję z pięciokątów

1. Zdefiniuj funkcję *pieciokat()* rysującą pięciokąt foremny o boku 50 pikseli.
2. Utwórz kompozycję podobną do przedstawionej na rysunku 15., wykorzystując funkcję *pieciokat()*.
3. Zapisz program w pliku pod nazwą *Pieciokaty*. Uruchom program.

Wskazówka: Taką samą kompozycję wykonywaliśmy w programie Scratch (temat 13.), tylko bez stosowania funkcji.

Warto zapamiętać

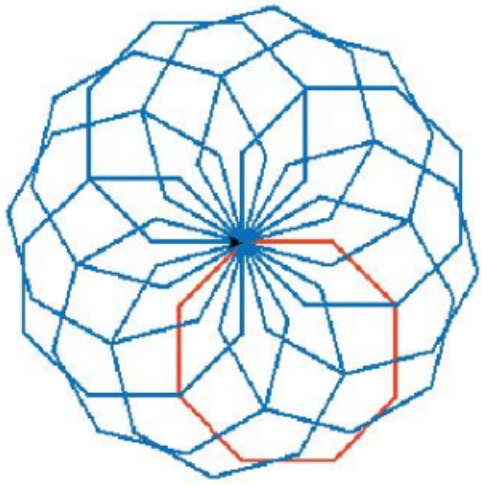
- Program w języku Python piszemy w edytorze kodu źródłowego, który jest częścią zintegrowanego środowiska programistycznego IDLE. Aby korzystać z IDLE, należy zainstalować darmową aplikację Python.
- IDLE zawiera (poza edytorem kodu źródłowego) powłokę Pythona i interpreter.
- Jeżeli w programie wystąpi błąd, interpreter przerwie wykonanie programu i wyświetli komunikat o błędzie, który może pojawiać się w osobnym oknie lub w oknie powłoki Pythona.
- Język Python posiada swój zbiór instrukcji, słów kluczowych i zasady składni oraz moduły z funkcjami, m.in. moduł `turtle` zawierający funkcje umożliwiające programowanie grafiki.
- Aby powstał obraz na ekranie, programujemy ruchy żółwia, np. przesuwanie do przodu, obroty.
- Powtarzające się polecenia można zaprogramować w języku Python, stosując instrukcję iteracyjną **`for`**.
- W języku Python można definiować własne funkcje z parametrami lub bez nich. Wywołanie funkcji w programie głównym powoduje wykonanie instrukcji będących treścią funkcji.

Pytania i polecenia

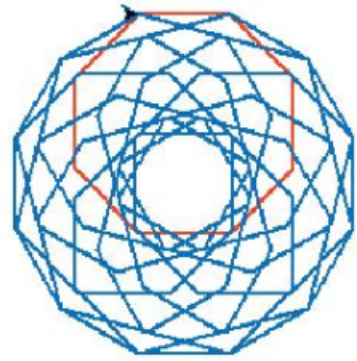
1. Co jest potrzebne, aby pisać programy w języku Python?
2. Omów na przykładzie etapy tworzenia programu w języku Python.
3. Na czym polega grafika żółwia w języku Python?
4. Podaj przykład zastosowania instrukcji **`for`** w programie w języku Python.
5. Dlaczego warto zdefiniować funkcję rysującą kwadrat, aby utworzyć złożoną kompozycję?
6. W którym miejscu programu powinno się umieścić definicję funkcji, a gdzie ją wywołać?
7. Czym jest wywołanie funkcji?

Zadania

1. Utwórz program rysujący dziewięciokąt foremny o boku 80 pikseli. Ustaw według własnego pomysłu kolory: pisaka, wypełnienia i tła. Zapisz program w pliku pod nazwą *Dziewieciokat_funkcja*.
2. Zdefiniuj funkcję *osmiokat()*, rysującą ośmiokąt foremny o boku 50 pikseli. Wywołaj funkcję w programie głównym. Dodaj kolor linii i zmień rozmiar pisaka według własnego pomysłu. Zapisz program w pliku pod nazwą *Osmiokat_funkcja*.
3. Otwórz plik *Osmiokat_funkcja* zapisany w zadaniu 2. Zmodyfikuj program, aby żółw rysował figurę podobną do pokazanej na rysunku 16a. Zapisz program w pliku pod nazwą *Osmiokaty1*.
4. Otwórz plik *Osmiokaty1* zapisany w zadaniu 3. Zmodyfikuj program, aby żółw rysował figurę podobną do pokazanej na rysunku 16b. Zapisz program w pliku pod nazwą *Osmiokaty2*.
   Wskazówka: W odpowiednim miejscu programu należy dodać polecenie:
   `turtle.forward(50)`.



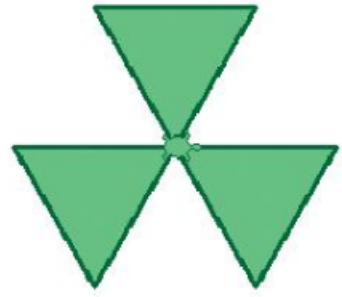
Rys. 16a. Kompozycja z ośmiokątów – zadanie 3.



Rys. 16b. Kompozycja z ośmiokątów – zadanie 4.

5. Korzystając z funkcji *kwadrat(bok)* zdefiniowanej w ćwiczeniu 7., napisz program rysujący pięć kwadratów w sposób pokazany na rysunku 17. Zapisz program w pliku pod nazwą *Kompozycja_5kwadratow*.
6. Zdefiniuj funkcję *trojkat()* rysującą trójkąt równoboczny o boku 100. Utwórz program rysujący figurę podobną do pokazanej na rysunku 18. Zapisz program w pliku pod nazwą *Koniczynka*.
7. Utwórz kompozycję z figur według własnego pomysłu, wykorzystując zdefiniowane w tym temacie funkcje.

Rys. 17. Pięć kwadratów – zadanie 5.



Rys. 18. Koniczynka – zadanie 6.

Dla zainteresowanych

8. Korzystając z funkcji *kwadrat(bok)* zdefiniowanej w ćwiczeniu 7., napisz program rysujący słup składający się z dziesięciu kwadratów, podobny do pokazanego na rysunku 19. Funkcję kwadrat wywołaj z parametrem 30. Dla kolejno rysowanych kwadratów zwiększaj o 1 piksel rozmiar pisaka. Zapisz program w pliku pod nazwą *Slup_z_kwadratow*.
   Wskazówki: Aby zwiększać rozmiar pisaka o 1 piksel, umieść w pętli `for` polecenie: `turtle.pensize(i+1)`. Kolor pokazany na rysunku 19. to fuksja (ang. *fuchsia*).

Rys. 19. Słup z kwadratów – zadanie 8.

9. Korzystając z funkcji *kwadrat(bok)* zdefiniowanej w ćwiczeniu 7. i funkcji *trojkat()* zdefiniowanej w zadaniu 6., utwórz trzy programy rysujące kompozycje podobne do pokazanych na rysunkach 20a-20c.

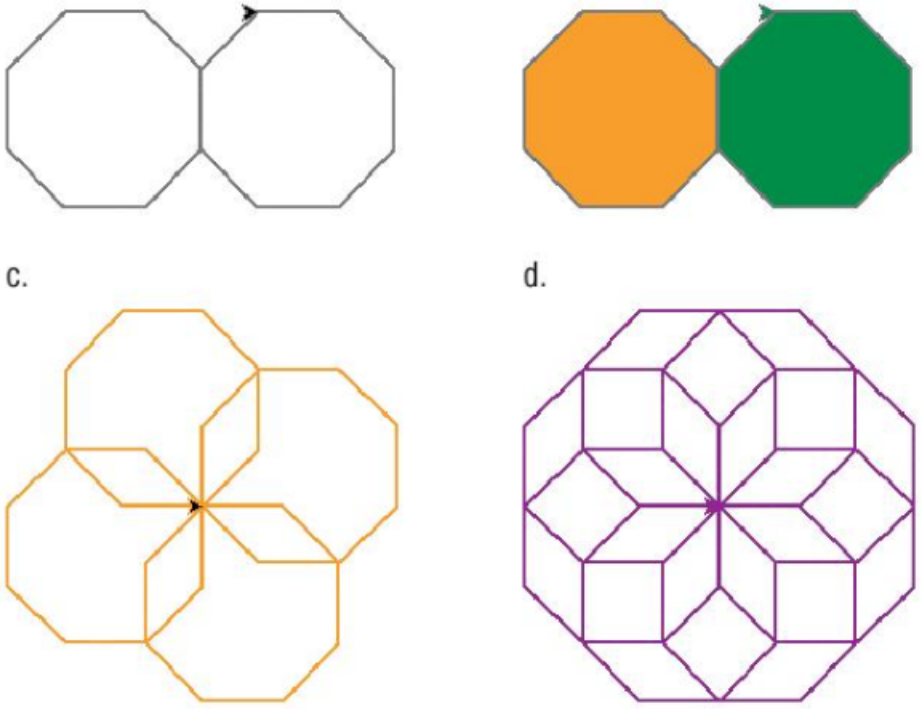
a. b. c.

Rys. 20. Kompozycje z kwadratów i trójkątów – zadanie 9.

10. Korzystając z funkcji *osmiokat()* zdefiniowanej w zadaniu 2., utwórz kompozycje podobne do pokazanych na rysunkach 21a-21d.

a. b.

c. d.



Rys. 21. Kompozycje z ośmiokątów – zadanie 10.