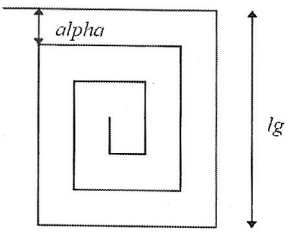
przekazywania elementów wyniku końcowego, dysponując nim, program nie ma potrzeby przekazywania wyniku obliczeń do góry, piętro po piętrze. Po prostu w momencie, w którym program „stwierdzi", że obliczenia zostały zakończone, procedura wywołująca zostanie o tym poinformowana wprost z ostatniego aktywnego poziomu rekurencji. Co za tym wszystkim idzie, nie ma absolutnie żadnej potrzeby zachowywania kontekstu poszczególnych poziomów pośrednich — liczy się tylko ostatni aktywny poziom, który dostarczy wynik, i basta!

Myślenie rekurencyjne

Pomimo oczywistych przykładów na to, że rekurencja jest dla człowieka czymś jak najbardziej naturalnym, niektórzy mają pewne trudności z używaniem jej podczas programowania. Nieumiejętność wyczucia istoty tej techniki programowania może wynikać z braku dobrych i poglądowych przykładów na jej wykorzystanie. Powodowany tym stwierdzeniem postanowiłem wybrać kilka prostych programów rekurencyjnych, generujących znane motywy graficzne — ich dobre zrozumienie nie będzie wystarczającym testem na oszacowanie zdolności myślenia rekurencyjnego (ale nawet wówczas wykonanie zadań zamieszczonych pod koniec rozdziału będzie jak najbardziej wskazane).

Przykład 1. Spirala

Zastanówmy się, jak rekurencyjnie narysować spiralę jednym pociągnięciem kreski (rysunek 3.7).



RYSUNEK 3.7.
Spirala narysowana rekurencyjnie

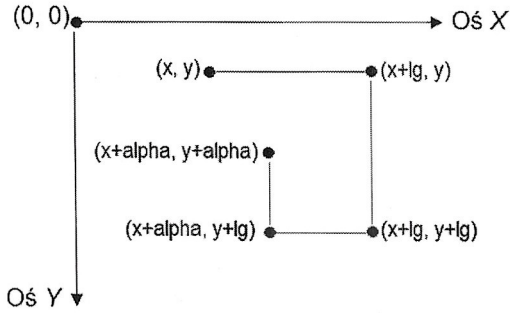
Parametrami programu są:

- odstęp pomiędzy liniami równoległymi — `alpha`;
- długość boku rysowanego w pierwszej kolejności — `lg`.

Algorytm iteracyjny byłby również nieskomplikowany (zwykła pętla), ale załóżmy, że chwilowo zapomnimy o jego istnieniu i wykonamy to samo rekurencyjnie. Istota rekurencji polega głównie na znalezieniu właściwej dekompozycji problemu. Tutaj jest ona przedstawiona na rysunku i w związku z tym ewentualne przetłumaczenie jej na program komputerowy powinno być znacznie ułatwione.

Rekurencyjność naszego zadania jest oczywista, gdyż program wynikowy zajmuje się powtarzaniem głównie tych samych czynności (rysuje linie poziome i pionowe, jednakże o różnej długości). Naszym zadaniem będzie odszukanie schematu rekurencyjnego i warunków zakończenia procesu wywołań rekurencyjnych.

Jak rozwiązać to zadanie? Najpierw przybliżmy się nieco do rzeczywistości ekranowej i wybierzmy jako punkt startowy pewien punkt o współrzędnych (x, y). Idea rozwiązania polega na narysowaniu czterech odcinków zewnętrznych spirali i dotarciu do punktu (x', y'). W tym nowym punkcie startowym możemy już wywołać rekurencyjnie procedurę rysowania, obarczoną oczywiście pewnymi warunkami gwarantującymi jej poprawne zakończenie (rysunek 3.8).



RYSUNEK 3.8. Spirala narysowana rekurencyjnie — analiza przypadku elementarnego

Program w Pythonie realizujący ten algorytm musi się odnieść oczywiście do określonych możliwości wybranej biblioteki graficznej. Jest ich wiele, ale na potrzeby tej książki wybrałem prostą realizację z użyciem modułu `tkinter`. Nie chcę w tym miejscu opisywać zasad jej użycia, gdyż nie pasuje to do koncepcji książki, ale w skrócie wymienię tylko zastosowane metody graficzne:

- Obiektem podstawowym biblioteki jest okno tworzone za pomocą wywołania komendy `Tk()`.
- Podstawowa metoda `geometry()` pozwala na ustalenie rozmiarów i pozycji na ekranie:

```
okno=Tk()
okno.geometry("width_size x height_size + x_position + y_position"):
# width_size : szerokość, height_size: wysokość
# x_position: przesunięcie poziome (w prawo), y_position: przesunięcie pionowe (w dół)
```

- W oknie możemy wbudowywać rozmaite widżety graficzne, zlecając ich rozstawienie tzw. menedżerowi geometrii — mamy tutaj kilka możliwości, ja wybrałem menedżera o nazwie `grid` (pol. *siatka*), który pozwala umieścić w wybranym oknie widżety, rysując je na umownej siatce składającej się z kolumn (`col`) i wierszy (`row`), w której lewa skrajna komórka ma współrzędne (0, 0), np. poniższy kod rysuje w pierwszej kolumnie i drugim wierszu etykietę `alpha=`:

```
napisAlpha = Label(okno, text=" alpha=")          # Etykieta 'alpha='
napisAlpha.grid(column=0, row=1, padx=5, pady=5)
```

- Podobnie możemy rozmieścić inne widżety: pole służące do wprowadzania danych (obiekt `Entry`), płótno do rysowania (`Canvas`) itp.
- W celu uzyskania prostej interakcji z użytkownikiem w kodzie znajdziesz także definicję kilku przycisków (widżet `Buton`), do których stworzyłem funkcje-handlery wywołujące określony kod po wciśnięciu (nazwę funkcji handlera wymieniasz w parametrze `command`). Przykład:

```
def clickedCzysc():
    # Tutaj jakieś instrukcje...
przyciskCzysc = Button(okno, text=" Czyść ", command=clickedCzysc)
```

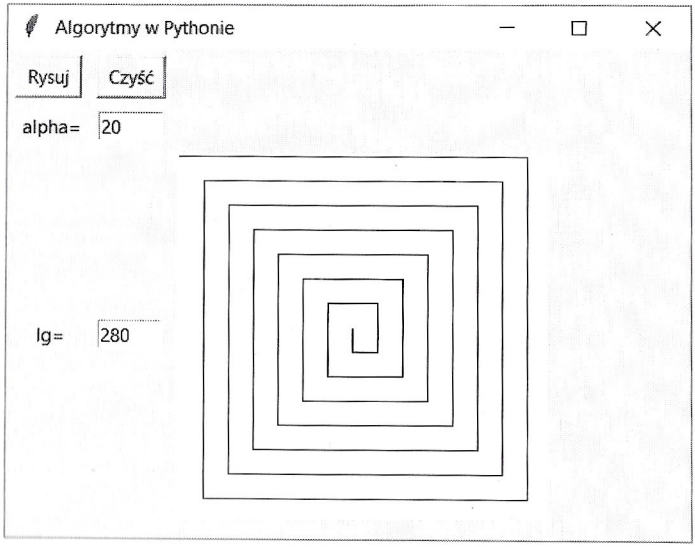
- Do samego rysowania na „płótnie" służy metoda `create_line(x1, y1, x2, y2)`, która powoduje narysowanie odcinka od punktu o współrzędnych (x_1, y_1) do (x_2, y_2) — moduł `tkinter` zakłada, że lewy górny róg płótna ma współrzędne (0, 0).

Jedna z kilku możliwych wersji programu, który realizuje koncepcję z rysunku 1.6, jest przedstawiona na listingu *spirala.py*.

```
from tkinter import *
okno=Tk()
okno.title(" Algorytmy w Pythonie ")
okno.geometry("550x400+500+60")

# Wartości domyślne:
paramAlpha=20
paramLg=180

napisAlpha = Label(okno, text=" alpha=")          # Etykieta 'alpha='
napisAlpha.grid(column=0, row=1, padx=5, pady=5) # Odstęp od brzegów komórki
editAlpha = Entry(okno, width=6)                  # Pole edycyjne 'Alpha'
editAlpha.insert(END, '20')          # Wstawiamy w pole wartość domyślną "20"
editAlpha.grid(column=1, row=1)      # Druga kolumna, drugi wiersz

napis2 = Label(okno, text=" lg=") # Etykieta 'lg='
napis2.grid(column=0, row=2)
editLg = Entry(okno, width=6)       # Pole edycyjne 'lg'
editLg.insert(END, '180')           # Wstawiamy wartość domyślną "180"
editLg.grid(column=1, row=2, padx=5, pady=5)

plotno = Canvas(bg="white", width=300, height=300)# Pole rysowania (płótno)

def clickedCzysc():
    editAlpha.delete(0, END)      # Czyści zawartość pola editAlpha
    editLg.delete(0, END)         # Czyści zawartość pola editLg
    plotno.delete("all")          # Czyści pole rysowania

przyciskCzysc = Button(okno, text=" Czyść ", command=clickedCzysc)
przyciskCzysc.grid(column=1, row=0, padx=5, pady=5)


def rysuj_spirala(paramAlpha, paramLg, x, y):
    if (paramLg > 0 and paramLg > paramAlpha):
        plotno.create_line(x, y, x + paramLg, y)
        plotno.create_line(x + paramLg, y, x + paramLg, y + paramLg)
        plotno.create_line(x + paramLg, y + paramLg, x + paramAlpha, y + paramLg)
```

```
        plotno.create_line(x + paramAlpha, y + paramLg, x + paramAlpha,
        ↪y + paramAlpha)
        rysuj_spirala(paramAlpha, paramLg - 2*paramAlpha, x+paramAlpha,
        ↪y + paramAlpha)
    plotno.grid(column=2, row=2, padx=1, pady=1)

def clickedRysuj():
    plotno.delete("all")                    # Czyści pole rysowania
    paramAlpha=int(editAlpha.get())         # Odczyt wartości Alpha
    paramLg=int(editLg.get())               # Odczyt wartości Lg
    print(paramLg, paramAlpha)
    rysuj_spirala(paramAlpha, paramLg, 5, 5) # Wywołanie algorytmu

przyciskRysuj = Button(okno, text=" Rysuj ", command=clickedRysuj)
przyciskRysuj.grid(column=0, row=0)
okno.mainloop() # Jeśli zamkniesz okno, to przejdziesz poza tę instrukcję
```

Przykładowe rozwiązanie w Pythonie zaprezentowano na rysunku 3.9.



RYSUNEK 3.9. Spirala narysowana rekurencyjnie — program uruchomiony w Windowsie

Program graficzny został przygotowany przy użyciu podstawowej biblioteki `tkinter`, ale nie jest to jedyna metoda programowania GUI w Pythonie i rezultat może wydać się nieco siermiężny. Jej zaletą jest jednak mały rozmiar kodu źródłowego, pozwalający uwypuklić samą istotę algorytmu!

Zwróć uwagę na warunek pozwalający na wywołanie rekurencyjne — błąd w tym miejscu może spowodować albo zapętlenie algorytmu, albo co najmniej uszkodzenie rysunku.