



**POLITECHNIKA LUBELSKA  
WYDZIAŁ ELEKTROTECHNIKI I  
INFORMATYKI**

**KIERUNEK STUDIÓW  
INFORMATYKA**

*MATERIAŁY DO ZAJĘĆ  
LABORATORYJNYCH*

**Wprowadzenie do informatyki**

Autor:  
mgr inż. Marek Kamiński

Lublin 2019

## **INFORMACJA O PRZEDMIOCIE**

### **Cele przedmiotu:**

Cel 1. Zapoznanie studentów z podstawowymi technikami programowania i podstawami algorytmizacji oraz wprowadzenie do podstawowych algorytmów i struktur danych.

Cel 2. Zapoznanie studentów z podstawowymi zagadnieniami teorii automatów i języków formalnych oraz maszyny Turinga.

Cel 3. Zapoznanie studentów z teorią złożoności obliczeniowej.

### **Efekty kształcenia w zakresie umiejętności:**

Efekt 1. Student potrafi opracować i zapisać w postaci schematu NS algorytmy dla prostych zadań programistycznych.

Efekt 2. Student potrafi używać prostych struktur danych.

Efekt 3. Student umie skonstruować automat dla prostego zadania tekstowego.

### **Literatura do zajęć:**

#### Literatura podstawowa

1. "Make Your Own Python Text Adventure", Phillip Johnson, Apress 2018.
2. "Program Arcade Games: With Python and Pygame, Fourth Edition", Paul Vincent Craven, Apress 2016.
3. "Beginning Programming with Python For Dummies, 2nd Edition", John Paul Mueller, John Wiley & Sons, Inc 2018.
4. "Schematy zwarte NS. Przykłady i zadania", Kwiatkowska A., Łukasik E, Mikom 2004.

#### Literatura uzupełniająca

1. "Beginning Python Games Development with Pygame", Harrison Kinsley and Will McGugan, Apress 2015.
2. "Invent your own computer games with python, 4th edition.", Al Sweigart, No Starch Press 2017.
3. "Head First Python, Second Edition", Paul Barry, O'Reilly 2017.
4. "Algorytmy i struktury danych = programy", Wirth N., WNT 1980.

### **Metody i kryteria oceny:**

#### Oceny cząstkowe:

- Ocena 1 Przygotowanie merytoryczne do zajęć laboratoryjnych na podstawie: wykładów, literatury, pytań kontrolnych do zajęć.
- Ocena 2 Zaliczenie pisemne z laboratorium (dwa kolokwia)

#### Ocena końcowa - zaliczenie przedmiotu:

- Pozytywne oceny cząstkowe.
- Ewentualne dodatkowe wymagania prowadzącego zajęcia.



**Fundusze Europejskie**  
Wiedza Edukacja Rozwój



**Rzeczpospolita Polska**



**Unia Europejska**  
Europejski Fundusz Społeczny

**Plan zajęć laboratoryjnych:**

Nr	Nazwa laboratorium	Wymiar
Lab1.	Algorytmizacja, schematy zwarte – programy z pętlami, programy używające tablic.	16 godz.
Lab2.	Procedury i funkcje, rekurencja.	4 godz.
Lab3.	Klasyczne algorytmy, wyszukiwanie i sortowanie.	4 godz.
Lab4.	Konstrukcja automatów skończonych dla wybranych zadań tekstowych (wyrażenia regularne).	2 godz.
Lab5.	Maszyna Turinga.	2 godz.
Lab6.	Teoria złożoności obliczeniowej.	2 godz.



**Fundusze Europejskie**  
Wiedza Edukacja Rozwój



**Rzeczpospolita  
Polska**

**Unia Europejska**  
Europejski Fundusz Społeczny



## **LABORATORIUM 1. ALGORYTMIZACJA, SCHEMATY ZWARTE – PROGRAMY Z PĘTLAMI, PROGRAMY UŻYWAJĄCE TABLIC.**

### **Cel laboratorium:**

Przedstawienie technik projektowania algorytmów. Omówienie budowy oraz zastosowania diagramów NS na potrzeby algorytmiki. Omówienie podstawowych struktur danych oraz metod ich przetwarzania.

### **Zakres tematyczny zajęć:**

1. Wprowadzenie do schematów zwartych NS oraz języka Python
  - a. Przedstawienie budowy schematu zwartego NS,
  - b. Przedstawienie zintegrowanego środowiska programistycznego (IDE) języka Python na przykładzie PyCharm,
  - c. Zapoznanie się z podstawowymi zasadami składniowymi języka Python,
  - d. Uruchomienie pierwszego skryptu.
2. Podstawowe operacje na danych. Interakcja z użytkownikiem. Instrukcja warunkowa
  - a. Przedstawienie podstawowych operacji na danych liczbowych, logicznych oraz tekstowych w języku Python.
  - b. Omówienie metod pobierania wartości od użytkownika oraz wyświetlenia wyników obliczeń na ekranie.
  - c. Przedstawienie składni oraz omówienie zastosowania instrukcji warunkowej.
  - d. Reprezentacja: operacji na danych, instrukcji wejścia/wyjścia oraz instrukcji warunkowej na schematach zwartych NS.
3. Wprowadzenie do pętli. Tryb pracy krokowej.
  - a. Omówienie budowy pętli oraz jej implementacji w języku Python.
  - b. Omówienie trybu pracy krokowej środowiska PyCharm (podgląd stanu zmiennych w trakcie działania programu, techniki lokalizowania błędów w kodzie).
  - c. Analiza działania aplikacji z użyciem trybu pracy krokowej.
  - d. Rozwiązywanie zadań z użyciem pętli.
4. Wprowadzenie do tablic
  - a. Omówienie budowy tablicy.
  - b. Omówienie podstawowych operacji na tablicach (oraz ich związku z pętlami).
  - c. Przedstawienie sposobu użycia tablicy na prostym przykładzie.
  - d. Rozwiązywanie zadań z użyciem pętli.
5. Powtórzenie materiału – rozwiązywanie zadań, analiza wybranych rozwiązań
  - a. Realizacja algorytmów związanych z użyciem instrukcji warunkowych i pętli
  - b. Rozwiązywanie zadań z zastosowaniem tablic
  - c. Analiza zamieszczonych diagramów NS
  - d. Przeniesienie rozwiązania w formie graficznej do postaci implementacyjnej.
6. Rozwiązywanie zadań do samodzielnej realizacji
  - a. Powtórzenie dotychczas zdobytej wiedzy oraz sprawdzenie swoich umiejętności programowania podczas rozwiązywania zadań.

### **Pytania kontrolne: Brak**



**Fundusze Europejskie**  
Wiedza Edukacja Rozwój



**Rzeczpospolita Polska**



**Unia Europejska**  
Europejski Fundusz Społeczny

## **ROZDZIAŁ 1. WPROWADZENIE DO SCHEMATÓW ZWARTYCH NS ORAZ JĘZYKA PYTHON**

### **Cel rozdziału:**

Omówienie budowy oraz zastosowania schematów zwartych NS w zakresie algorytmiki. Przedstawienie wybranego środowiska pracy języka Python.

### **Zakres tematyczny:**

1. Przedstawienie budowy schematu zwartego NS,
2. Przedstawienie zintegrowanego środowiska programistycznego (IDE) języka Python na przykładzie PyCharm,
3. Zapoznanie się z podstawowymi zasadami składniowymi języka Python,
4. Uruchomienie pierwszego skryptu.

### **Pytania kontrolne:**

Brak

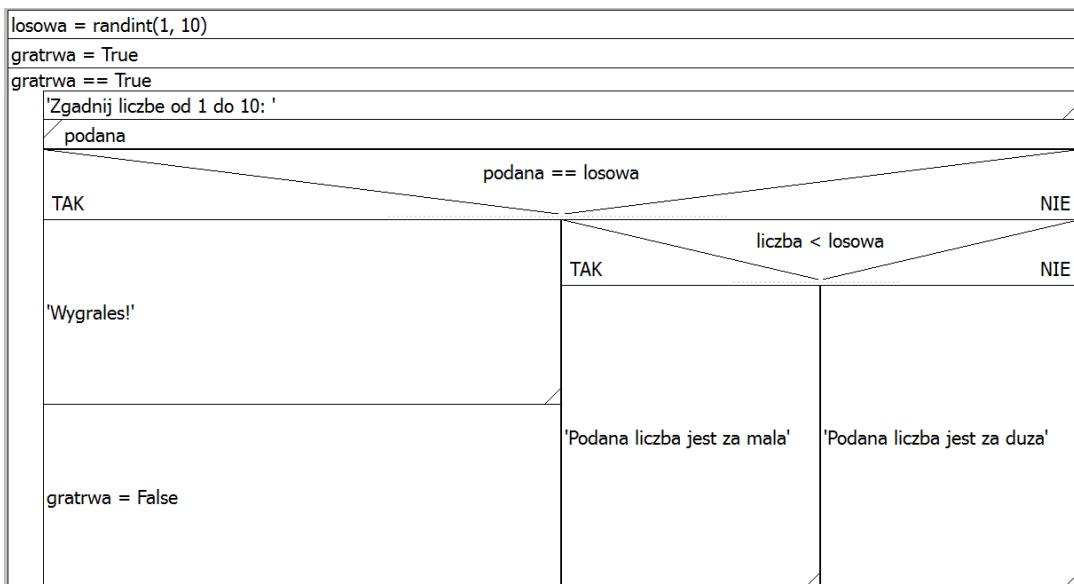
### **1. Wprowadzenie**

Główym celem niniejszego laboratorium jest wyrobienie u czytelnika zdolności algorytmicznego myślenia – zdolności, która pozwoli na szybsze odnajdywanie zależności zarówno w świecie wszechobecnej techniki komputerowej, jak i codziennym życiu. Wyrobienie tego typu myślenia wymaga sporego nakładu czasu – czasu znacznie przekraczającego ograniczony czas spotkań w ramach tego laboratorium. Zazwyczaj najlepsze efekty przynosi znalezienie sobie własnego projektu – gry lub aplikacji, której rozwijanie będzie motywowało do stałego poszerzania swojej wiedzy oraz umiejętności.

### **2. Schematy zwarte NS**

W celu wizualizacji i lepszego zrozumienia zasady działania programów komputerowych, stosunkowo często (szczególnie na początku) wykorzystywana jest reprezentacja graficzna, gdzie każda składowa programu reprezentowana jest jako oddzielny bloczek (najczęściej w postaci prostej figury geometrycznej). Podobne techniki stosuje się również z dużych firmach podczas prac nad rozbudowanymi projektami programistycznymi, gdzie reprezentacja graficzna pozwala skupić się na działaniu algorytmu bez wnikania w szczegóły technologiczne.

Na potrzeby niniejszego laboratorium wybrano metodę wizualizacji przy użyciu schematów zwartych NS. Przykładowy diagram przedstawiony został na rys. 1.1.



Rys. 1.1. Diagram NS przykładowego programu

Znaczenie poszczególnych bloczków składających się na diagram NS będzie sukcesywnie wprowadzane na kolejnych laboratoriach. Na ten moment należy wiedzieć, iż schemat działania programu wykonywany jest zawsze od góry do dołu – podobnie jak kod programu w klasycznym języku programowania.

#### Kącik autora

Czy domyślasz się, jakie zadanie realizują poszczególne bloczki przedstawione na rysunku? Jakie zadanie realizuje powyższy kod?

W powyższym przykładzie wykorzystano między innymi:

- instrukcję przypisania (w pierwszej instrukcji dodatkowo wywołano funkcję),
- pętlę (wielokrotne wykonanie sekwencji instrukcji),
- instrukcję wyjścia (wypisanie wiadomości),
- instrukcję wejścia (pobranie informacji),
- oraz instrukcję warunkową (sterowanie pracą programu).

Każdy z wymienionych elementów reprezentowany jest przez bloczek o nieco innym kształcie – ciekawostką może stanowić fakt, iż na powyższym diagramie użyte zostały niemalże wszystkie dostępne typy bloczków. Oznacza to, że dysponując jedynie pięcioma typami bloczków jest się w stanie opisać praktycznie każdy algorytm, grę, czy aplikację!



Fundusze  
Europejskie  
Wiedza Edukacja Rozwój



Rzeczpospolita  
Polska



Unia Europejska  
Europejski Fundusz Społeczny

### 3. Język programowania Python

Po zapoznaniu się z działaniem algorytmu w formie graficznej, przychodzi moment wprowadzenia sekwencji instrukcji do komputera – zadanie to realizowane jest najczęściej poprzez napisanie instrukcji w konkretnym języku programowania. Spośród wielu dostępnych języków na potrzeby laboratorium wybrano zdobywający coraz większą popularność język Python, który dzięki swojej przyjaznej składni idealnie nadaje się na rozpoczęcie przygody z programowaniem.

Wraz ze wzrostem mocy komputerów, coraz częściej podczas pracy z kodem wykorzystywane są tzw. zintegrowane środowiska graficzne (z ang. IDE), które znacząco usprawniają pracę programisty, oferując między innymi:

- szybką modyfikację kodu źródłowego,
- natychmiastowe sprawdzenie efektów pracy programu,
- kolorowanie oraz formatowanie kodu,
- pracę krokową (wykonywanie instrukcji po instrukcji),
- autozupełnianie kodu (wystarczy wpisać pierwsze litery, aby uzyskać odpowiedź),
- wykrycie błędów składniowych,
- wykrycie niektórych błędów logicznych,
- oraz wiele, wiele innych funkcji.

Kod może zostać napisany zasadniczo w dowolnym edytorze tekstowym, by później przekazać go do komplikacji i uruchomienia bez udziału środowiska graficznego, jednak takie podejście znacząco wydłuża proces uruchomienia aplikacji i lokalizacji błędów.

W związku z powyższym, zdecydowano się na wybór jednego z darmowych i ogólnodostępnych środowisk języka Python, istniejącego pod nazwą PyCharm. W dalszej części skryptu przedstawiony zostanie proces instalacji oraz konfiguracji środowiska z poziomu systemu Windows (środowisko dostępne jest również w wersjach pod systemy: Linux oraz macOS)

W przypadku systemu operacyjnego Windows, w celu korzystania ze środowiska należy:

1. Pobrać oraz zainstalować środowisko uruchomieniowe języka Python ze oficjalnej strony internetowej: <https://www.python.org/downloads/>
2. Pobrać oraz zainstalować środowisko PyCharm ze strony internetowej pod adresem: <https://www.jetbrains.com/pycharm/download/> (wersja Community jest wersją w pełni darmową),
3. Uruchomić ponownie komputer w celu aktualizacji ścieżek systemowych,



Fundusze Europejskie  
Wiedza Edukacja Rozwój



Rzeczpospolita  
Polska



Unia Europejska  
Europejski Fundusz Społeczny

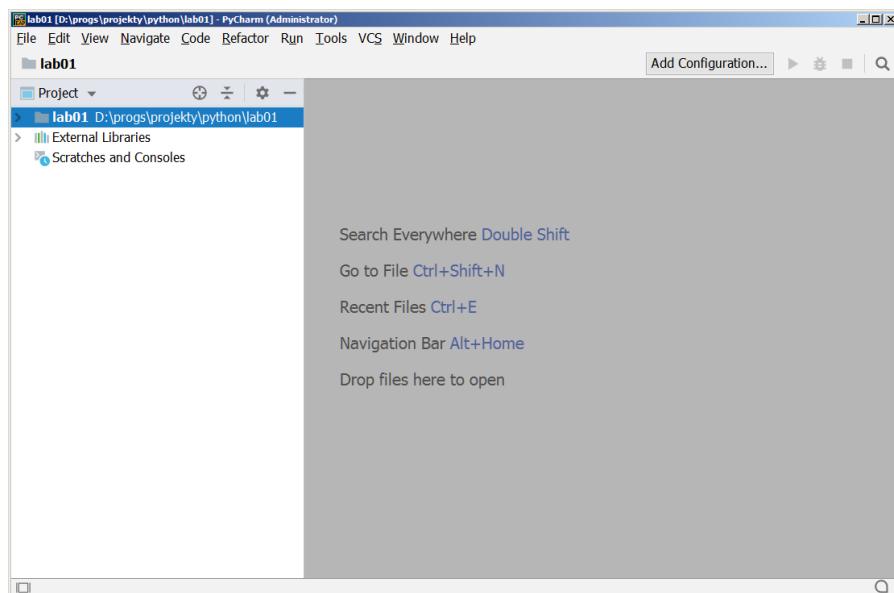
4. W systemie odnaleźć skrót do środowiska PyCharm.

Po wykonaniu powyższych czynności przed oczami użytkownika powinno ukazać się okno wyglądem zbliżone do tego przedstawionego na rys. 1.2.



Rys. 1.2. Okno powitalne środowiska PyCharm

Od tego momentu możliwe jest tworzenie własnych projektów w języku Python. Do utworzenia projektu („Create New Project”) niezbędne jest zazwyczaj jedynie podanie jego nazwy – środowisko PyCharm powinno automatycznie odnaleźć pobrane i zainstalowane w pierwszym kroku środowisko uruchomieniowe języka Python. Efektem utworzenia projektu jest okno widoczne na rys. 1.3.



Rys. 1.3. Okno projektu środowiska PyCharm

Domyślnie projekt nie zawiera żadnego pliku pozwalającego na wprowadzenie kodu źródłowego – plik ten należy utworzyć samodzielnie.

## Zadanie 1

Utwórz projekt o dowolnej nazwie, a następnie dodaj do niego plik źródłowy z kodem aplikacji zamieszczonym na rys. 1.4.

```
1  from random import *
2
3  losowa = randint(1, 10)
4  gratrwa = True
5
6  while gratrwa:
7      podana = int(input('Zgadnij liczbę od 1 do 10: '))
8
9      if podana == losowa:
10          print('Wygrałeś!')
11          gratrwa = False
12      elif podana < losowa:
13          print('Podana liczba jest za mała')
14      else:
15          print('Podana liczba jest za duża')
16
```

Rys. 1.4. Kod pierwszego programu w języku Python

### Kącik autora

Na tym etapie nie musisz znać przeznaczenia wszystkich instrukcji – sprawdź, czy uda Ci się uruchomić program i zobacz, jak wygląda komunikacja aplikacji z użytkownikiem.

## Realizacja zadania

Nowy plik źródłowy można utworzyć na kilka różnych sposobów, w tym poprzez:

1. Menu główne środowiska PyCharm z użyciem sekwencji:  
File -> New... -> Python File
2. Kliknięcie prawym przyciskiem myszy na nazwę aktywnego projektu, a następnie:  
New -> Python File
3. Użycie skrótu klawiszowego alt+insert, a następnie wybranie opcji Python File

Po każdej z powyższych czynności należy podać nazwę nowego pliku źródłowego. Przyjęła się konwencja nazewnicza, w której plik od którego rozpoczyna się wykonywanie programu w obrębie projektu przyjmuje nazwę „main” (z ang. plik główny) – w ten sposób jest znacznie łatwiej zlokalizować miejsce rozpoczęcia programu (szczególnie w przypadku projektów składających się z setek lub nawet tysięcy plików z kodem źródłowym). Samo nazewnictwo zależy jednak od konkretnego języka (np. w języku PHP oraz HTML za stronę startową przyjmuje się najczęściej plik o nazwie index, odpowiednio: index.php lub index.html)



Fundusze Europejskie  
Wiedza Edukacja Rozwój

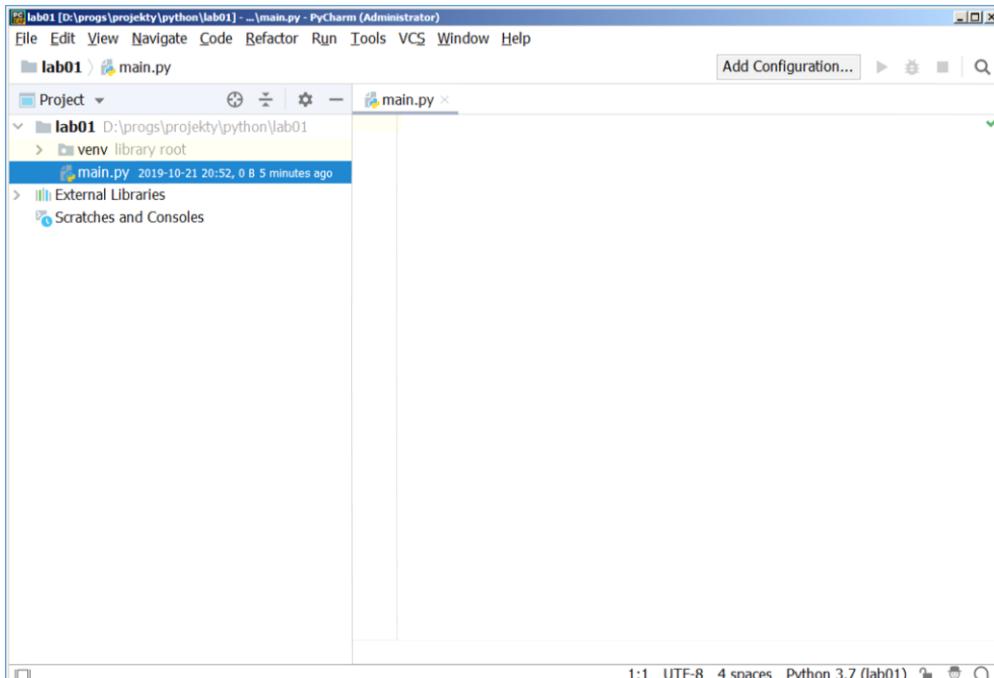


Rzeczpospolita  
Polska



Unia Europejska  
Europejski Fundusz Społeczny

W związku z powyższym, pierwszemu plikowi można z powodzeniem przypisać nazwę „main”. Mimo, że pliki z kodem języka Python mają zazwyczaj rozszerzenie „.py”, rozszerzenia nie trzeba bezpośrednio wpisywać – środowisko doda je automatycznie podczas tworzenia pliku. W efekcie powinno się ukazać okno środowiska z otworzonym plikiem „main.py” przedstawione na rys. 1.5.



Rys. 1.5. Utworzenie pliku main.py

Aby uruchomić skrypt należy odnaleźć plik „main.py” w bocznym oknie projektu, kliknąć na niego prawym przyciskiem myszy i wybrać opcję „Run main”. Od tego momentu plik „main.py” zostanie wybrany, jako początkowy plik projektu, co pozwoli na przyspieszenie kolejnych uruchomień – aktywny stanie się zarówno zielony przycisk „run” widoczny obok przycisku „Add Configuration”, jak i skrót klawiszowy „shift+F10”.



Fundusze  
Europejskie  
Wiedza Edukacja Rozwój

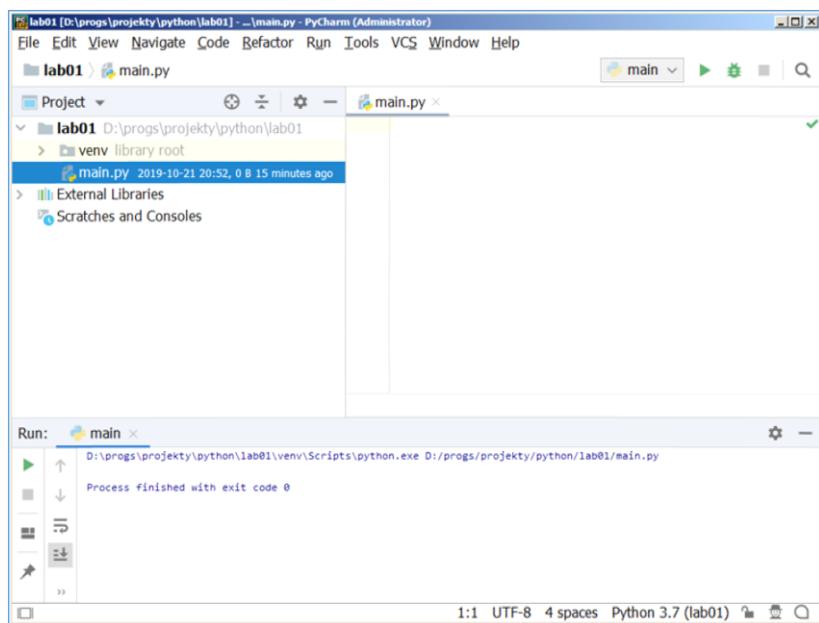


Rzeczpospolita  
Polska



Unia Europejska  
Europejski Fundusz Społeczny

## Zintegrowany Program Rozwoju Politechniki Lubelskiej – część druga



Rys. 1.6. Efekt uruchomienia pustego skryptu

Do tak utworzonego pliku należy wprowadzić kod przedstawiony na rys. 1.4 z dokładnym zachowaniem formatowania oraz wielkości poszczególnych liter – język Python wymaga również odwzorowania wszystkich widocznych na rysunku wcięć (standardowo o długości czterech spacji). Po przepisaniu całości należy ponownie uruchomić skrypt (poprzez kliknięcie ikony lub aktywację skrótu klawiszowego)

A screenshot of the PyCharm IDE interface. The title bar says "lab01 [D:\progs\projekty\python\lab01] - main.py - PyCharm (Administrator)". The project tree on the left shows a "lab01" folder containing a "venv" folder and a "main.py" file. The code editor window shows the "main.py" file with the following code:

```
from random import *
losowa = randint(1, 10)
gratrwa = True

while gratrwa:
    podana = int(input('Zgadnij liczbe od 1 do 10: '))

    if podana == losowa:
        print('Wygrales!')
        gratrwa = False
    elif podana < losowa:
        print('Podana liczba jest za mala')
    else:
        print('Podana liczba jest za duza')
```

The "Run" tool window shows the command "D:\progs\projekty\python\lab01\venv\Scripts\python.exe D:/progs/projekty/python/lab01/main.py" and the user input "Zgadnij liczbe od 1 do 10: 5". The status bar at the bottom indicates "1:1 UTF-8 4 spaces Python 3.7 (lab01)".

Rys. 1.7. Efekt uruchomienia przykładowu

W przypadku wprowadzenia kodu bez błędów, wykonywanie programu powinno rozpoczęć się we wbudowanej w środowisko konsoli (standardowo umiejscowionej pod oknem z kodem źródłowym), która odpowiadać będzie za interakcję użytkownika z aplikacją (rys. 1.7).



Fundusze  
Europejskie  
Wiedza Edukacja Rozwój



Rzeczpospolita  
Polska

Unia Europejska  
Europejski Fundusz Społeczny



Wystąpienie nawet jednego błędu nie pozwoli na uruchomienie programu – w takim przypadku środowisko postara się wskazać miejsce wystąpienia błędu wraz z jego krótkim opisem słownym.

Przykładowe błędy, które można spotkać przy wprowadzaniu powyższego kodu:

- niezachowanie odpowiednich wcięć – język Python wymaga, aby bloki kodu posiadały wcięcie,
- niezachowanie wielkości liter – instrukcja „gratrwa = true” będzie traktowana jako błąd (w przeciwnieństwie do poprawnej wersji „gratrwa = True”),
- niedomknięcie nawiasów lub cudzysłowu,
- pominięcie symbolu dwukropka w przypadku pętli oraz instrukcji warunkowych (problem ten zostanie poruszony w dalszej części laboratorium).

Niekiedy wskazanie środowiska nie jest w pełni jednoznaczne i szczególnie na początku przygody z programowaniem potrzeba nieco więcej czasu na lokalizację i usunięcie błędu – przykładowo niedomknięcie nawiasu powoduje jednokrotnie podświetlenie pięciu linii kodu sugerując wystąpienie błędu w każdej z nich (rys. 1.8).

```
from random import *
losowa = randint(1, 10)
gratrwa = True

while gratrwa:
    podana = int(input('Zgadnij liczbe od 1 do 10: '))
    if podana == losowa:
        print('Wygrales!')
        gratrwa = False
    elif podana < losowa:
        print('Podana liczba jest za mala')
    else:
        print('Podana liczba jest za duza')

while gratrwa > if podana == losowa
```

Rys. 1.8. Efekt niedomknięcia jednego nawiasu

Uruchomienie przykładowej aplikacji kończy wstęp do programowania w języku Python – w następnych rozdziałach omówione zostaną poszczególne składowe kodu oraz techniki ich łączenia w większe aplikacje.



Fundusze  
Europejskie  
Wiedza Edukacja Rozwój



Rzeczpospolita  
Polska

Unia Europejska  
Europejski Fundusz Społeczny



## ROZDZIAŁ 2. PODSTAWOWE OPERACJE NA DANYCH. INTERAKCJA Z UŻYTKOWNIKIEM. INSTRUKCJA WARUNKOWA

### Cel rozdziału:

Przedstawienie podstawowych operacji na danych w języku Python wraz z omówieniem metod pobierania wartości od użytkownika oraz wyświetlenia wyników obliczeń na ekranie. Wprowadzenie do instrukcji warunkowej.

### Zakres tematyczny:

1. Przedstawienie podstawowych operacji na danych liczbowych, logicznych oraz tekstowych w języku Python.
2. Omówienie metod pobierania wartości od użytkownika oraz wyświetlenia wyników obliczeń na ekranie.
3. Przedstawienie składni oraz omówienie zastosowania instrukcji warunkowej.
4. Reprezentacja: operacji na danych, instrukcji wejścia/wyjścia oraz instrukcji warunkowej na schematach zwartych NS.

### Pytania kontrolne:

1. Jak wygląda struktura diagramu NS?
2. Jak utworzyć nowy projekt w środowisku PyCharm?

### 1. Przechowywanie danych – pojęcie zmiennej

Zmienną nazywa się zarezerwowane miejsce w pamięci przechowujące pewną wartość. W kodzie programu, do zmiennej można się odwołać za pośrednictwem wybranej w momencie jej tworzenia nazwy (ważne jest, aby zmiennym przypisywać nazwy, które możliwe jak najlepiej opisują przechowywane przez nie dane). W momencie tworzenia zmiennej wybierany jest również zazwyczaj typ danych przechowywanych przez zmienną – do podstawowych typów danych zaliczamy między innymi: typ tekstowy, liczby: całkowite i zmiennoprzecinkowe oraz typ logiczny (prawda-fałsz) – rys. 2.1.



Rys. 2.1. Zmienne przechowujące różne typy danych



Fundusze Europejskie  
Wiedza Edukacja Rozwój



Rzeczpospolita  
Polska



## 2. Dane liczbowe

Po utworzeniu zmiennej i przypisaniu jej wartości początkowej programista otrzymuje możliwość wykonywania z jej udziałem szeregu operacji. Zakres operacji na zmiennych zależy w głównej mierze od ich typu – w tym rozdziale przedstawione zostaną operacje, które dotyczą wartości liczbowych. Zgodnie z rys. 2.1. dostępne są dwa główne typy liczbowe: typ całkowity (int) oraz typ zmiennoprzecinkowy (float).

Wśród dostępnych operacji możemy wyróżnić:

1. przypisanie wartości liczbowej (lub innej zmiennej) do zmiennej

```
wzrost = 180  
wzrost_drugiej_osoby = wzrost
```

2. zamianę dwóch wartości (operacja dostępna również dla wszystkich innych typów)

```
wzrost1 = 180 # liczba całkowita  
wzrost2 = 160  
  
(wzrost1,wzrost2) = (wzrost2,wzrost1)
```

3. podstawowe operacje matematyczne (dodawanie, odejmowanie, mnożenie, dzielenie)

```
waga_pierwszej_osoby = 70.3 #liczba zmiennoprzecinkowa  
waga_drugiej_osoby = 55.8  
  
wspolna_waga = waga_pierwszej_osoby + waga_drugiej_osoby  
roznica_wagi = waga_pierwszej_osoby - waga_drugiej_osoby  
srednia_waga = wspolna_waga / 2  
  
limit_osob_w_windzie = 12  
dopuszczalna_waga_windy = srednia_waga * limit_osob_w_windzie
```

4. podstawowe operacje matematyczne (dodawanie, odejmowanie, mnożenie, dzielenie) operujące bezpośrednio na zmiennej znajdującej się po lewej stronie

```
wzrost = 160  
wzrost += 10      # wzrost = wzrost+10
```

5. resztę z dzielenia (modulo)

```
liczba_kawalkow_tortu = 16  
liczba_studentow = 5  
  
ile_zostanie_kawalkow_gdy_wszyscy_zjedza_po_rowno =  
liczba_kawalkow_tortu % liczba_studentow
```



Fundusze  
Europejskie  
Wiedza Edukacja Rozwój



Rzeczpospolita  
Polska



Unia Europejska  
Europejski Fundusz Społeczny

## 6. potęgowanie

```
podstawa_systemu_dwojkowego = 2  
liczba_bitow_na_bajt = 8  
  
ile_wartosci_moze_przyjac_jeden_bajt =  
    podstawa_systemu_dwojkowego ** liczba_bitow_na_bajt
```

Oczywiście powyższe operacje można ze sobą dowolnie łączyć. Kolejność wykonywania opiera się na zasadach znanych z matematyki – aby wymusić wykonanie jednego działania przed innym należy skorzystać z grupowania w nawiasy okrągłe:

```
wzrost1 = 170  
wzrost2 = 180  
  
srednia = (wzrost1 + wzrost2) / 2  
# zepsuta_srednia = wzrost1 + wzrost2 / 2
```

Niezastosowanie nawiasów doprowadziłoby do podzielenia zawartości zmiennej wzrost2 przez dwa, a następnie dodaniu jej do wartości zmiennej wzrost1 – w rezultacie, otrzymany wynik przyjąłby postać 260, zamiast prawidłowej wartości 175.

## 3. Dane logiczne (prawda-fałsz)

Typ logiczny oferuje możliwość przypisania do zmiennej jedynie dwóch wartości: True (prawda) oraz False (fałsz). Zmienne typu logicznego wykorzystywane są najczęściej razem z instrukcjami warunkowymi oraz pętlami, które zostaną omówione w dalszej części.

## 4. Dane tekstowe

Na potrzeby niniejszego laboratorium zostanie jedynie niewielka część możliwości, jakie oferuje język Python w kontekście pracy z danymi tekstowymi. Do definiowania wartości tekstowej może zostać użyty zarówno pojedynczy, jak i podwójny cudzysłów. Do podstawowej funkcjonalności można zaliczyć:

### 1. łączenie dwóch obiektów tekstowych

```
pierwsza_czesc = 'Pierwsza czesc wiadomosci'  
caly_komunikat = pierwsza_czesc + ' oraz druga czesc!'
```

### 2. łączenie obiektu tekstowego z danymi liczbowymi (zamiana liczby na tekst)

```
waga_uzytkownika = 68  
komunikat = "Uzytkownik wazy: " + str(waga_uzytkownika) + " kg."
```

### 3. zamiana tekstu na liczbę

```
wzrost_tekstowo = '178'  
wzrost_liczbowo = int(wzrost_tekstowo)  
waga_tekstowo = '68.5'  
waga_liczbowo = float(waga_tekstowo)
```

## 5. Interakcja z użytkownikiem

Na początku istnienia komputerów, ich rola ograniczała się głównie do wykonywania skomplikowanych obliczeń matematycznych, a sama interakcja z użytkownikiem była mocno ograniczona – pierwsze komputery nie posiadały dedykowanego monitora, ani klawiatury. Większość programów wykonywanych była w oparciu o taśmy z wygenerowanych wcześniej kodem (dziurka na taśmie oznaczała 0, jej brak – 1) i w taki sam sposób przedstawiane były wyniki. Z biegiem czasu komputery zaczęły wykonywać coraz więcej funkcji, przez co niezbędne stało się dodanie urządzeń pozwalających na interakcję z użytkownikiem – przez długi czas najpopularniejszym zestawieniem była klawiatura z monitorem. Idea myszki została wprowadzona znacznie później (jej szczyt popularności przypadł na okres wprowadzenia graficznych systemów operacyjnych).

Zasadniczo, w języku Python wyróżnić można dwie instrukcje odpowiadające za interakcję z użytkownikiem: `print` oraz `input`.

```
imie_uzytkownika = input('Podaj swoje imię: ')
waga_uzytkownika = int(input('Podaj swoją wagę: '))

print('Masz na imię ' + imie_uzytkownika + ' i ważysz ' +
      str(waga_uzytkownika) + ' kg.')
```

W obu przypadkach pierwszy parametr oznacza tekst, który zostanie wyświetlony w oknie konsoli. Funkcja `input` zwraca wartość pobraną od użytkownika, stąd niezbędne jest jej przypisanie do konkretnej zmiennej. Funkcja `print` nie zwraca żadnej wartości.

## 6. Instrukcja warunkowa

Ostatnim elementem przedstawionym w tym laboratorium są instrukcje warunkowe – używane są wszędzie tam, gdzie niezbędne jest podjęcie akcji w zależności od stanu wybranej zmiennej. Przykładowo, pobierając wzrost użytkownika aplikacja powinna m.in. sprawdzić, czy podana przez użytkownika liczba nie jest ujemna. Podobnie sytuacja wygląda w przypadku gier, czy aplikacji okienkowych – program musi nieustannie analizować wykonane przez użytkownika akcje (wyjście z aplikacji, naciśnięcie klawisza odpowiedzialnego za strzał) i odpowiednio na nie reagować (zupełnie inny klawisz odpowiada za ruch gracza w grze, niż za wyjście z rozgrywki).

W języku Python instrukcję warunkową tworzą trzy główne sekcje:

1. sekcja sprawdzenia warunku głównego (występuje zawsze),
2. sekcja warunków pobocznych (opcjonalna) – w przypadku niespełnienia warunku głównego sprawdzić można inne przypadki,
3. sekcja wykonania kodu w momencie, gdy żaden z powyższych warunków nie został spełniony.



Fundusze  
Europejskie  
Wiedza Edukacja Rozwój



Rzeczpospolita  
Polska



Unia Europejska  
Europejski Fundusz Społeczny

**Kącik autora**

Oddzielny mechanizm stanowi mechanizm obsługi wyjątków (np. w przypadku podania przez użytkownika tekstu w miejsce liczby), który omówiony zostanie przy innej okazji.

Praktyczne użycie instrukcji warunkowej zostało przedstawione na poniższym przykładzie (rys. 2.2). Program po wylosowaniu liczby od 1 do 20 prosi dwóch graczy o próbę odgadnięcia liczby, a następnie sprawdza, który z graczy był bliżej prawidłowej odpowiedzi, wyświetlając stosowny komunikat.

```
1  from random import *
2  from math import *
3
4  losowa = randint(1,20)
5  gracz1 = int(input('gracz1) Zgadnij liczbę od 1 do 20: '))
6  gracz2 = int(input('gracz2) Zgadnij liczbę od 1 do 20: '))
7
8  odleglosc1 = fabs(losowa - gracz1) #obliczenie wartości bezwzględnej
9  odleglosc2 = fabs(losowa - gracz2)
10
11 print('Wylosowana liczba wynosi: ' + str(losowa))
12
13 if odleglosc1 < odleglosc2:
14     print('Wygrał gracz nr1!')
15 elif odleglosc1 > odleglosc2:
16     print('Wygrał gracz nr2!')
17 else:
18     print('Remis!')
```

Rys. 2.2. Program wykorzystujący instrukcję warunkową

W przykładzie wywołana została funkcja `fabs` z modułu matematycznego języka Python (stąd w drugiej linii znalazła się zapis informujący o dołączeniu funkcji matematycznych), która zwraca wartość bezwzględną przekazanej wartości.

Po obliczeniu odległości od wylosowanej liczby dla każdego z graczy można sprawdzić, któremu z nich udało się trafić bliżej – w instrukcji warunkowej wykorzystano wszystkie trzy sekcje jednocześnie: `if`, `elif` oraz `else`.

W trakcie porównywania dwóch wartości wykorzystywane są operatory. W powyższym przykładzie zostały użyte dwa z nich: operator mniejszości oraz większości. Wśród najczęściej występujących operatorów można wyróżnić:

1. Operator równości (`==`) – podwójny znak równości używany jest do sprawdzenia, czy dwie wartości są takie same,
2. Operator nierówności (`!=`) – wykrzyknik ze znakiem równości używany jest do sprawdzenia, czy dwie wartości są od siebie różne,



Fundusze Europejskie  
Wiedza Edukacja Rozwój



Rzeczypospolita  
Polska



3. Operatory większości oraz mniejszości ( $>$  oraz  $<$ ) – sprawdzenie, czy wartość po lewej stronie porównania jest odpowiednio: większa, bądź mniejsza, niż wartość znajdująca się po prawej stronie.
4. Operatory większe-równe oraz mniejsze-równe ( $\geq$  oraz  $\leq$ ) – analogicznie do operatorów większości i mniejszości z tą różnicą, że warunek jest również spełniony w przypadku wystąpienia dwóch takich samych wartości po obu stronach porównania

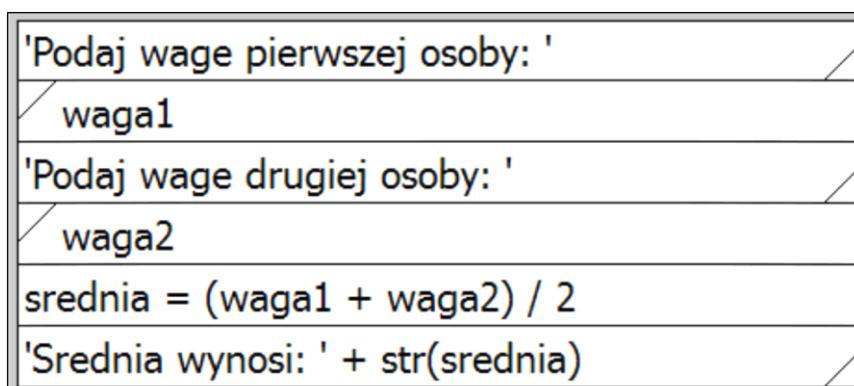
Dość często wykonanie akcji zależnych jest od sprawdzenia kilku warunków jednocześnie – składnia instrukcji warunkowej pozwala na taki zabieg z użyciem dodatkowych operatorów:

1. Operator i (`and`) – warunek jest spełniony, jeżeli wszystkie warunki są spełnione  
(np. `x >= 5 and x <= 15` – czy zmienna x znajduje się w przedziale 5-15?)
2. Operator lub (`or`) – warunek jest spełniony, gdy choć jeden z warunków jest spełniony  
(np. `x == 0 or x > 18` – czy zmienna x jest równa 0 lub jest większa od 18?)
3. Operator nie (`not`) – odwraca stan logiczny przyjętego wyrażenia  
(np. `not(wiek < 18)` – sprawdzenie, czy osoba jest pełnoletnia)

Omówienie operatorów kończy sekcję poświęconą instrukcji warunkowej.

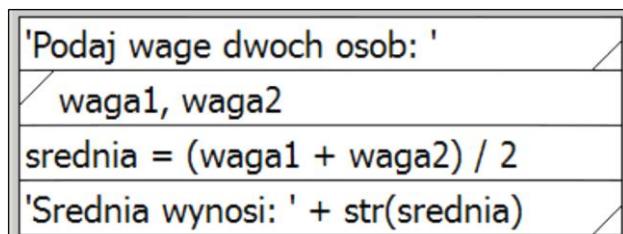
## 7. Reprezentacja na schematach zwartych NS

W obrębie diagramów NS, wszystkie operacje na zmiennych (podstawienia, operacje arytmetyczne) wykonywane są w bloczkach reprezentowanych przez pusty prostokąt. Dwa oddzielne typy bloczków dedykowane są operacjom: wejścia (wczytania) oraz wyjścia (wypisania). Te pierwsze zakończone są małym trójkątem w lewym górnym rogu, podczas gdy operacje wypisania posiadają mały trójkąt w prawym dolnym rogu, zgodnie z przykładem na rys. 2.3.



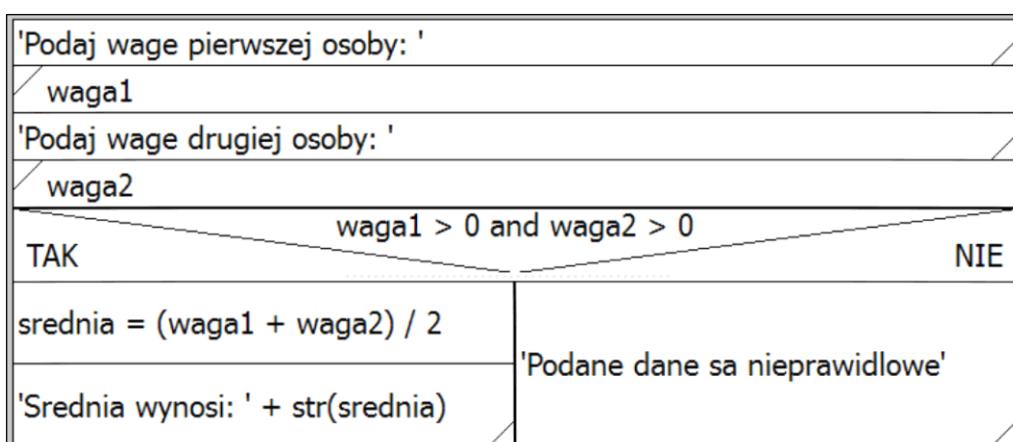
Rys. 2.3. Reprezentacja operacji: podstawienia, wczytania oraz wypisania na diagramie NS

Należy zwrócić uwagę na fakt, iż bloczek pobierający dane od użytkownika na diagramie NS nie odpowiada za wypisanie tekstu – w tym celu stosowany jest odrębny bloczek z operacją wypisania komunikatu. Notacja diagramów NS pozwala natomiast na wczytanie kilku wartości jednocześnie z poziomu jednego bloczku (rys. 2.4).



Rys. 2.4. Pobranie dwóch wartości od użytkownika w jednym bloczku

W przypadku chęci dodatkowego sprawdzenia danych wprowadzonych przez użytkownika, niezbędne jest dostawienie sekcji zawierającej instrukcję warunkową. W diagramach NS instrukcja warunkowa podzielona jest na: sekcję sprawdzenia warunku, sekcję wykonania kodu w przypadku warunku spełnionego oraz sekcję wykonania kodu w przypadku jego niespełnienia – sytuację tą przedstawiono na rys. 2.5.



Rys. 2.5. Reprezentacja instrukcji warunkowej na diagramie NS

W przypadku chęci dodania sekcji `elif` języka Python należy dodać kolejne instrukcje warunkowe w gałęzi "NIE" znajdującej się po prawej stronie diagramu.



**Zadania do samodzielnej realizacji (rozpisać diagram, a następnie zaimplementować)**

1. Zaprogramować jeden przebieg gry „Papier, nożyce, kamień”. W programie zawrzeć:
  - wypisanie wiadomość z prośbą o podanie jednego z symboli (0 – papier, 1 – nożyce, 2 – kamień),
  - wczytanie wartości od pierwszego gracza,
  - powtórzenie wypisania wiadomości oraz wczytania symbolu dla drugiego gracza,
  - sprawdzenie poprawności wprowadzonych danych,
  - wypisanie komunikatu o stanie pojedynku (wygrana gracza pierwszego, drugiego lub remis).
2. Pobrać od użytkownika trzy dodatnie liczby, a następnie odnaleźć i wypisać największą z nich.
3. Pobrać od użytkownika liczbę w zakresie 0 – 999 (w przypadku niespełnionego warunku wypisać stosowny komunikat). W programie wyświetlić sumę cyfr oraz liczbę: jedności, dziesiątek i setek składających się na podaną liczbę (podpowiedź: w zadaniu wykorzystać operator reszty z dzielenia)

Przykład:

Liczba na wejściu: 850

Wyjście: "Suma cyfr: 13, setki: 8, dziesiątki: 5, jedności: 0"

4. Opracować algorytm oraz zaimplementować program znajdujący oraz wyświetlający pierwiastki równania kwadratowego (do obliczenia pierwiastka z delty wykorzystać funkcję sqrt() biblioteki math)

Dla przypomnienia:

opis równania kwadratowego:  $ax^2+bx+c=0$

wyróżnik równania (delta):  $\Delta=b^2 - 4ac$

dla  $\Delta < 0$ , brak pierwiastków

$$\text{dla } \Delta = 0, x_0 = \frac{-b}{2a}$$

$$\text{dla } \Delta > 0, x_1 = \frac{-b-\sqrt{\Delta}}{2a} \text{ oraz } x_2 = \frac{-b+\sqrt{\Delta}}{2a}$$



Fundusze Europejskie  
Wiedza Edukacja Rozwój



Rzeczpospolita  
Polska



Unia Europejska  
Europejski Fundusz Społeczny

## **ROZDZIAŁ 3. WPROWADZENIE DO PĘTLI. TRYB PRACY KROKOWEJ.**

### **Cel rozdziału:**

Przedstawienie pojęcia pętli oraz metod rozwiązywania problemów z jej pomocą. Nauka korzystania z trybu pracy krokowej środowiska PyCharm.

### **Zakres tematyczny:**

1. Omówienie budowy pętli oraz jej implementacji w języku Python.
2. Omówienie trybu pracy krokowej środowiska PyCharm (podgląd stanu zmiennych w trakcie działania programu, techniki lokalizowania błędów w kodzie).
3. Analiza działania aplikacji z użyciem trybu pracy krokowej.
4. Rozwiązywanie zadań z użyciem pętli.

### **Pytania kontrolne:**

1. Jakie można wyróżnić typy danych w języku Python?
2. Jak zamienić tekst na liczbę?
3. W jaki sposób pobrać od użytkownika liczbę?
4. W jaki sposób wyświetlić w tej samej linii tekst oraz liczbę?
5. W jaki sposób można sprawdzić kilka warunków jednocześnie? Jaki operatory są w tym celu wykorzystywane?
6. Z jakich sekcji składa się instrukcja warunkowa?

### **1. Pętla – spektrum zastosowań**

W obecnych czasach, większość aplikacji uruchamianych w systemie pracuje w trybie ciągłym – to użytkownik zazwyczaj decyduje, kiedy chce zamknąć aplikację. Aby możliwa była ciągła interakcja z użytkownikiem niezbędne jest użycie konstrukcji, która pozwoli na nieprzerwane wyświetlanie stanu aplikacji przy jednoczesnym sprawdzaniu danych wprowadzonych przez użytkownika. W grach komputerowych przyjęło się nawet mówić o klatkach na sekundę, które oznaczają ile razy w ciągu sekundy zostanie wyświetlony obraz. Powyższe zadania realizowane są z użyciem pętli.

### **2. Rodzaje pętli w języku Python**

Jako, że język Python miał być z założenia językiem niezwykle przyjaznym początkującemu programiście, zdecydowano się na pozostawienie jedynie dwóch typów pętli (w innych językach programowania widuje się nawet cztery rodzaje). Pierwszym rodzajem pętli jest pętla ogólnego przeznaczenia `while`, której użycie będzie wystarczające dla większości problemów poruszanych w ramach niniejszego laboratorium. Przykładowe użycie pętli zamieszczono poniżej na rys. 3.1.



**Fundusze Europejskie**  
Wiedza Edukacja Rozwój



**Rzeczpospolita Polska**



```
1  ile_gier = int(input('Podaj liczbę kupionych gier: '))
2
3  if ile_gier > 0:
4      suma = 0
5      licznik = 0
6
7  while licznik < ile_gier:
8      cena = int(input('Podaj cenę kolejnej gry: '))
9      suma += cena
10
11     licznik += 1
12
13 srednia_cena = suma / ile_gier
14 print('Średnia cena pojedynczej gry: ' + str(srednia_cena))
```

Rys. 3.1. Przykład użycia pętli

Jak można zauważyc, pętla jest w pewnym sensie rozbudowaną wersją instrukcji warunkowej, gdzie na podobnej zasadzie sprawdzany jest warunek i w przypadku jego spełnienia wykonywany jest wyznaczony wcięciem blok kodu. Przy budowaniu pętli należy zwrócić uwagę na to, aby zapewnić możliwość wyjścia programu z pętli. Konsekwencją nawet małego błędu w logice może być tzw. pętla nieskończona – taką aplikację po uruchomieniu będzie trzeba zakończyć ręcznie z poziomu środowiska.

**Kącik autora**

Być może miałeś okazję doświadczyć sytuacji, w której podczas korzystania z gry lub aplikacji wyświetlony został monit systemu operacyjnego mówiący o braku odpowiedzi aplikacji i konieczności jej zamknięcia? Istnieje szansa, że do sytuacji tej doprowadziła właśnie pętla nieskończona!

Drugim rodzajem pętli jest znana również z innych języków pętla `for-each` (w języku Python nie występuje bezpośredni odpowiednik pętli `for` znanej z innych języków programowania). Użycie pętli `for-each` upraszcza korzystanie z list oraz tablic, jednak konstrukcja ta nie będzie powszechnie używana w ramach laboratorium.

Dodatkowym argumentem jest fakt, iż diagramy NS również wykorzystują jedynie pętlę `while`. W celu przedstawienia pętli z poziomu diagramu NS należy skorzystać z konstrukcji przedstawionej na rys. 3.2.

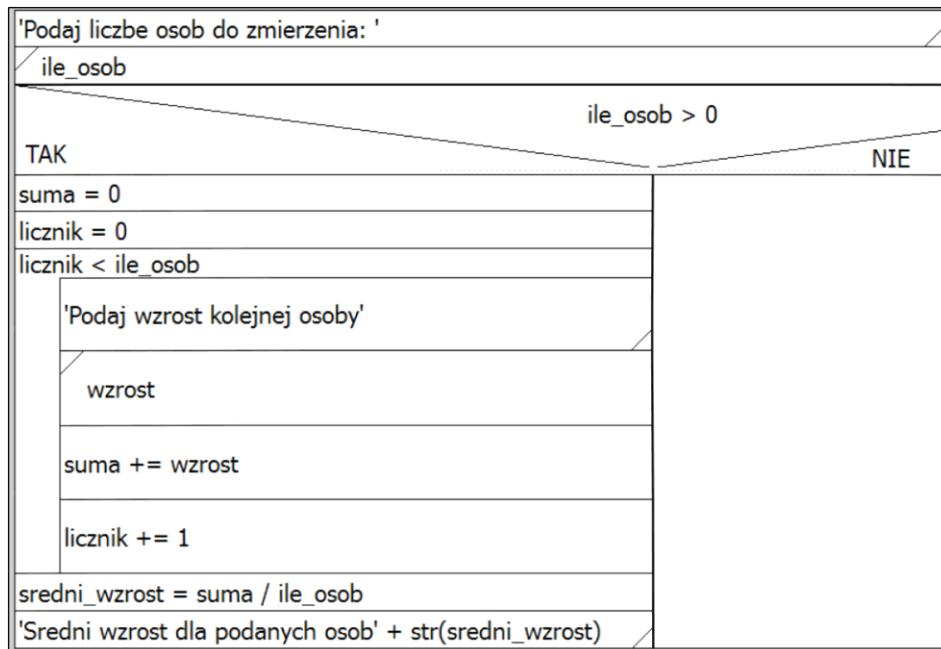


Fundusze  
Europejskie  
Wiedza Edukacja Rozwój



Rzeczpospolita  
Polska





Rys. 3.2. Reprezentacja pętli na diagramie NS

W konstrukcji pętli na diagramie można wyróżnić:

- sekcję warunku znajdująca się w lewym górnym rogu bloczka. Warunek jest sprawdzany zawsze przed wykonaniem kolejnego cyklu pętli.
- sekcję kodu do wykonania w przypadku spełnienia warunku. Blok kodu przyjmuje postać wciętego prostokąta, w którym można osadzać kolejne elementy diagramu.

#### Kącik autora

I to wszystko, co jest potrzebne do korzystania z pętli!

### 3. Praca krokowa w środowisku PyCharm

Przy pracy z petlami dość często zachodzi sprawdzenia stanu poszczególnych zmiennych w kolejnych cyklach pętli. Zadanie to można zrealizować przy pomocy funkcji `print` wyświetlając każdorazowo stan interesujących programistę zmiennych – jest to jednak rozwiążanie dość niepraktyczne ze względu na znaczne zmniejszenie czytelności kodu.

Z pomocą przychodzi specjalny tryb środowiska PyCharm (tryb ten jest również standardowo dostępny również w innych środowiskach) zwany trybem pracy krokowej. Idea pracy w tym trybie polega na:

1. określaniu, przy których instrukcjach środowisko będzie tymczasowo zatrzymywało wykonywanie programu,

2. po zatrzymaniu się programu w określonym miejscu, można odczytać z poziomu środowiska aktualny stan wszystkich zmiennych będących w użyciu (idealne narzędzie do analizy działania pętli oraz wyszukiwania ewentualnych błędów logicznych),
3. po odczytaniu stanu zmiennych możliwe jest wznowienie pracy programu do kolejnego miejsca wstrzymania lub do kolejnej linii – to programista decyduje, kiedy zostanie wykonany kolejny krok (stąd nazwa trybu pracy krokowej).

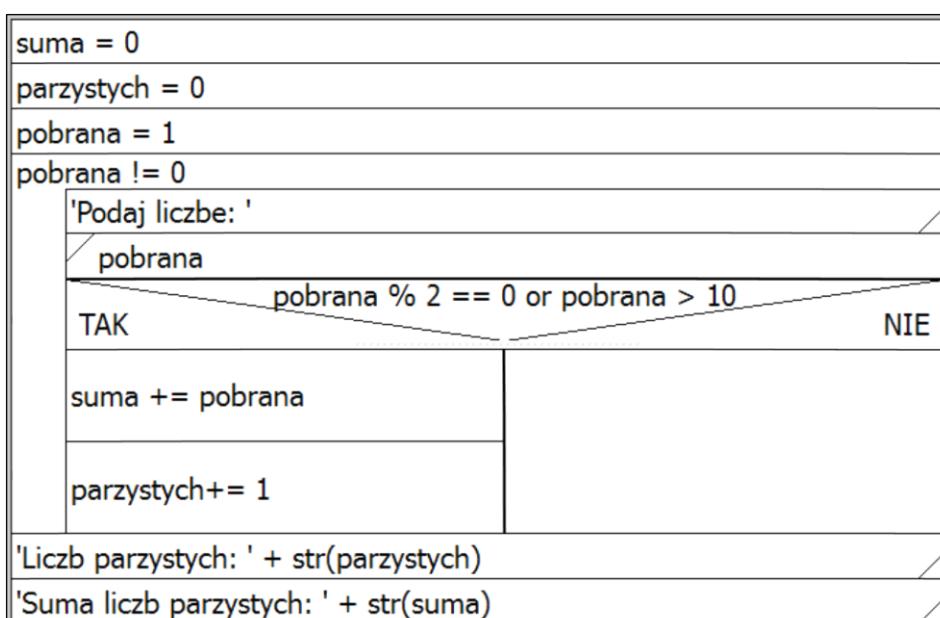
Pracę z trybem najlepiej będzie zaprezentować na konkretnym przykładzie.

### Przykładowe zadanie

W pętli pobierać od użytkownika liczby do momentu wpisania liczby zero. Po wyjściu w pętli wyświetlić ilość oraz sumę liczb parzystych większych od 10. Przygotować diagram NS oraz zaimplementować algorytm w języku Python.

### Rozwiążanie:

Zgodnie z treścią przygotowany został diagram NS widoczny na rys. 3.3.



Rys. 3.3. Rozwiążanie zadania z błędem logicznym

### Kącik autora

Czy potrafisz natychmiast odnaleźć błąd?

W oparciu o diagram widoczny na rys. 3.3, programista przystąpił do implementacji algorytmu w środowisku PyCharm. Efektem działania prac jest widoczny na rys. 3.4 kod źródłowy.

```
1  suma = 0
2  parzystych = 0
3
4  pobrana = 1
5  while pobrana != 0:
6      pobrana = int(input('Podaj liczbe: '))
7
8  if pobrana % 2 == 0 or pobrana > 10:
9      suma += pobrana
10     parzystych += 1
11
12 print('Liczba parzystych: ' + str(parzystych))
13 print('Suma liczb parzystych: ' + str(suma))
14
```

Rys. 3.4. Rozwiązywanie zadania z błędem logicznym – język Python

Po implementacji programista postanowił sprawdzić efekt działania swojego programu kilkukrotnie uruchamiając program podając różne dane wejściowe. Efekt pierwszego uruchomienia został zaprezentowany poniżej:

```
Podaj liczbe: 12
Podaj liczbe: 18
Podaj liczbe: 20
Podaj liczbe: 7
Podaj liczbe: 0

Liczba parzystych: 4
Suma liczb parzystych: 50
```

Przy pierwszym uruchomieniu suma liczb parzystych zgadza się z treścią oraz wymogami zadania ( $12+18+20$ ), jednak już tutaj widać, że algorytm naliczył cztery liczby większe od 10, podczas gdy rzeczywiście podano tylko trzy liczby spełniające ten warunek.

Dalsze testowanie programu utwierdziło programistę o błędzie w kodzie:

```
Podaj liczbe: 15
Podaj liczbe: 15
Podaj liczbe: 0

Liczba parzystych: 3
Suma liczb parzystych: 30
```

oraz test programu dla kilku liczb mniejszych od 10:

```
Podaj liczbe: 2
Podaj liczbe: 4
Podaj liczbe: 8
Podaj liczbe: 0

Liczba parzystych: 4
Suma liczb parzystych: 14
```

Ostatecznie żaden powyższy przypadek nie spełniał wymogów realizowanego zadania.



Fundusze  
Europejskie  
Wiedza Edukacja Rozwój



Rzeczpospolita  
Polska



Unia Europejska  
Europejski Fundusz Społeczny

## Analiza programu z użyciem pracy krokowej

W celu zlokalizowania błędu można by było skorzystać z instrukcji `print` wypisując stan zmiennych po każdej linii, jednak doprowadziłoby to do zmniejszenia czytelności kodu oraz potrzeby dodawania i usuwania wielokrotnych wywołań instrukcji `print` za każdym razem, gdy w kodzie pojawiłby się trudny do zlokalizowania błąd.

W zaprezentowanym przykładzie, miejscem najbardziej narażonym na błąd logiczny (co przekłada się bezpośrednio na błędny wynik) jest operacja warunkowa wewnętrz pętli `while`. Programista przy analizie błędu chciałby skorzystać z możliwości podglądu stanu zmiennych przy każdym obrocie pętli. Aby skorzystać z tej możliwości należy kliknąć lewym przyciskiem myszy w obszar znajdujący się między numer linii, a kodem źródłowym – kliknięcie w pusty obszar spowoduje wyświetlenie czerwonej kropki (tzw. punktu przerwania) widocznej na rys. 3.5.

```
1 suma = 0
2 parzystych = 0
3
4 pobrana = 1
5 while pobrana != 0:
6     pobrana = int(input('Podaj liczbe: '))
7
8     if pobrana % 2 == 0 or pobrana > 10:
9         suma += pobrana
10    ● parzystych += 1
11
12 ● print('Liczba parzystych: ' + str(parzystych))
13 print('Suma liczb parzystych: ' + str(suma))
```

Rys. 3.5. Zaznaczenie miejsc, w których zatrzyma się program podczas pracy krokowej

Punkt ten można ustawić jedynie w linii, w której znajduje się już jakaś instrukcja (nie może to być pusta linia) – przykładowo, punktu nie można ustawić w linii o numerze 11.

Do tej pory napisane aplikacje uruchamiane były za pośrednictwem przycisku „Run” lub skrótu klawiszowego (dla przypomnienia, domyślnie jest to shift+F10). Uruchomienie aplikacji w trybie pracy krokowej odbywa się analogicznie – przy ikonie przycisku „Run” znajduje się ikona reprezentująca robaka (z ang. debug), która uruchamia pracę krokową. Alternatywną metodą jest użycie skrótu shift+F9.

Po wejściu w tryb, początkowy efekt uruchomienia programu nie różni się zbytnio od trybu pracy standardowej (użytkownik proszony jest o podanie pierwszej wartości), jednak po wprowadzeniu liczby, aplikacja zostaje wstrzymana, a programista zostaje przeniesiony do okna podglądu zmiennych (rys. 3.6).



Fundusze  
Europejskie  
Wiedza Edukacja Rozwój



Rzeczpospolita  
Polska



Unia Europejska  
Europejski Fundusz Społeczny

The screenshot shows a code editor with Python code and a debugger window below it. The code is as follows:

```

9     suma += pobrana
10    parzystych += 1
11
12  ● print('Liczba parzystych: ' + str(parzystych))
13  print('Suma liczb parzystych: ' + str(suma))
14
15

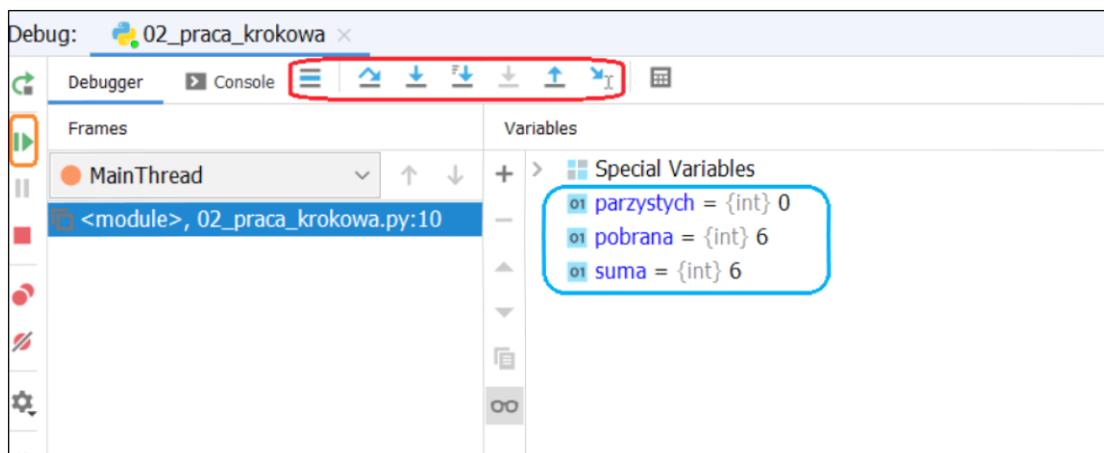
```

The debugger window shows the current frame is MainThread, and the line being executed is 12. The variables pane shows:

- parzystych = {int} 0
- pobrana = {int} 6
- suma = {int} 6

Rys. 3.6. Zaznaczenie miejsc, w których zatrzyma się program podczas pracy krokowej

Okno podglądu zmiennych przedstawia stan zmiennych przed wywołaniem wskazanej instrukcji. Najważniejsze elementy panelu pracy krokowej przedstawiono na rys. 3.7.



Rys. 3.7. Podgląd stanu zmiennych oraz panel sterujący trybem pracy krokowej

W niebieskiej ramce umieszczono sekcję podglądu stanu zmiennych – zawarto w nim informację o typie zmiennej oraz aktualnie przyjętej wartości. Po przeanalizowaniu wartości zmiennych programista może wznowić pracę programu jednym z przycisków znajdujących się w sekcji sterującej oznaczonej czerwoną ramką. W przypadku, gdy programistę interesuje jedynie stan aplikacji w punktach przerwania, programista może wykorzystać przycisk „Resume Program” (zaznaczony pomarańczową ramką, skrót F9), by kontynuować pracę z programem.

W większości przypadków dla pracy w trybie wykorzystywana będzie druga ikona od lewej („Step Over”, dostępna pod skrótem F8), która pozwoli na przejście do kolejnej linii kodu. W momencie pracy z funkcjami oraz wieloma plikami na znaczeniu zyskają również



pozostałe opcje, ze szczególnym uwzględnieniem opcji trzeciej („Step Into”, dostępna pod skrótem F7), która pozwoli m.in. na wejście do ciała funkcji.

Jeżeli programista chciałby przeanalizować działanie kodu od samego początku, z powodzeniem można ustawić punkt przerwania w pierwszej linii i następnie kontynuować wykonywanie programu już tylko z użyciem skrótu F8. Istotną zaletą tego rozwiązania jest fakt, iż na bieżąco można obserwować stan zmiennych, również z poziomu okna z kodem źródłowym – środowisko wyświetla aktualny stan zmiennych tuż przy ich nazwie (rys. 3.8).

```
1 ● suma = 0    suma: 0
2     parzystych = 0  parzystych: 0
3
4     pobrana = 1  pobrana: 5
5     while pobrana != 0:
6         pobrana = int(input('Podaj liczbe: '))
7
8         if pobrana % 2 == 0 or pobrana > 10:
9             suma += pobrana
10            parzystych += 1
11
12     print('Liczba parzystych: ' + str(parzystych))
13     print('Suma liczb parzystych: ' + str(suma))
```

Rys. 3.8. Analiza stanu zmiennych z poziomu kodu źródłowego

Niezależnie od wybranego rozwiązania, wprowadzenie liczby spełniającej warunek instrukcji warunkowej spowoduje wykonanie zawartego w niej kodu. Na rys. 3.7. przedstawiono stan programu po jednym przebiegu pętli, gdzie użytkownik podał wartość równą 5. Fakt ten sugeruje, że wystąpił błąd logiczny w trakcie budowy instrukcji warunkowej – warunki w niej umieszczone są prawidłowe, jednak użyto błędnego operatora do ich połączenia. Po zmianie operatora z „or” na „and” program zacznie działać zgodnie z zamierzeniem.

### Zadanie do realizacji

Sprawdzić działanie poprawionej wersji programu z użyciem trybu pracy krokowej.



Fundusze  
Europejskie  
Wiedza Edukacja Rozwój



Rzeczpospolita  
Polska

Unia Europejska  
Europejski Fundusz Społeczny



### Zadania do samodzielnej realizacji [klasyczne]

1. Posiadając wiedzę na temat pętli, przygotować diagram oraz zaimplementować grę „Papier, nożyce, kamień”. Program kończy swoją pracę w przypadku wygranej jednego z graczy – w przypadku remisu należy powtórzyć rozgrywkę.
2. Wczytać w pętli n liczb z przedziału od 10 do 50 (wartość n pobrać od użytkownika przed wejściem do pętli), a następnie wyświetlić sumę ich kwadratów.
3. Wczytywać liczby od użytkownika do momentu, gdy ich suma przekroczy 255 lub iloczyn będzie większy od 50000, a następnie wyświetlić ich średnią arytmetyczną.
4. Pobierać od użytkownika wartości do momentu podania liczby nieparzystej. Wyznaczyć najmniejszą i największą wartość, a następnie wyświetlić wynik potęgowania (operację potęgowania zrealizować w oparciu o pętlę, bez wykorzystania wbudowanej funkcji), gdzie: podstawa = min, potęga = max.

### Zadania do samodzielnej realizacji [tekstowe]

Wskazówki do zadań tekstowych:

1. Większość zadań tekstowych korzysta z liczb pseudolosowych – najprostszym sposobem jest wykorzystanie funkcji `randint(a,b)` z biblioteki `math` (gdzie: `a` oznacza zakres dolny losowania, podczas gdy `b` górny). Przykładowo, w celu wylosowania liczby od 1 do 10 można skorzystać z następującej instrukcji:

```
liczba = randint(1, 10)
```

2. Przed rozpisaniem diagramu NS warto się nieco dłużej zastanowić nad rozmieszczeniem i wielkością bloczków – zazwyczaj największym bloczkiem jest główna pętla programu.
3. Aby zapytać gracza o wybór, najprościej jest użyć obudowanej instrukcji `input()` w następującej postaci:

```
wybor = int(input('Wybierz akcję (0 - wyjście, 1 - akcja): '))
```



Fundusze Europejskie  
Wiedza Edukacja Rozwój



Rzeczpospolita  
Polska



Unia Europejska  
Europejski Fundusz Społeczny

## 1. Gra planszowa

Dana jest plansza postaci:

...a.....b....e....c..d

01234567890123456789

gdzie:

- a) idziesz pięć pól do przodu,
- b) czekasz od 1 do 3 tur,
- c) wracasz na start,
- d) wygrywasz grę,
- e) wpadłeś w pułapkę i tracisz turę. W kolejnych turach musisz uzyskać na kostce cztery oczka, aby wyrwać się z pułapki – w przeciwnym razie pozostajesz dalej w pułapce.

Implementacja:

- w każdej turze rzucić kostką czterościenną (liczba od 1 do 4 – można zmienić zasady w przypadku wydłużenia mapy) i przenieść gracza o wybraną liczbę pozycji,
- podjąć akcję stosowną do pola, na którym znajduje się gracz,
- sprawdzić warunki wygranej,
- wyświetlić informacje o wygranej gracza o danym numerze (bądź po prostu gracza w przypadku gry jednoosobowej).

Dodatkowe informacje:

- planszę można wydłużyć i rozszerzyć dodatkowe pola (obecnie bez użycia tablic, choć ich zastosowanie mogłoby znacznie powiększyć możliwości generowania mapy i zautomatyzować jej obsługę),
- należy podjąć decyzję, czy plansza jest „zapętlona” – w przypadku, gdy nie jest, nietrafienie przez gracza idealnie w pole „d” spowoduje ponowne przejście przez start, zgodnie z liczbą wyrzuconych oczek.



Fundusze  
Europejskie  
Wiedza Edukacja Rozwój



Rzeczpospolita  
Polska



Unia Europejska  
Europejski Fundusz Społeczny

## 2. Oczko

Oczko – prosta gra karciana, polegająca na dobieraniu kolejnych kart dotąd, aby osiągnąć wartość liczbową posiadanych kart jak najbliższą (ale nie większą niż) 21. Gracz otrzymuje kolejne karty z talii dotąd, aż sam zdecyduje, że nie chce już więcej kart, lub otrzyma wynik 21 lub większy. Suma większa lub równa 22 oznacza przegrana. Wyjątkiem od tej reguły jest perskie oczko (dwa asy). Perskie oczko zawsze oznacza wygraną. W oczko gra się talią od 2 do asa.

Punktacja:

- Karty 2 do 10 mają wartość równą wartości karty
- Figury:
  - walet – 2 pkt,
  - dama – 3 pkt,
  - król – 4 pkt,
  - as – 11 pkt.

(źródło opisu: wikipedia)

Implementacja zakłada udział jedynie dwóch graczy. Po wejściu do pętli głównej programu, każdy gracz decyduje o tym, czy chce dociągnąć dodatkową kartę – w momencie, gdy zechce, losowana jest liczba od 2 do 14, gdzie:

- wylosowanie liczb od 2 do 10 ma wartość od 2 do 10,
- wylosowanie liczb od 11 do 13 (walet, dama, król) ma odpowiednio wartości: 2, 3 oraz 4,
- wylosowanie liczby 14 daje wartość 11. [\*]

Gracz, który zrezygnuje z rozgrywki, bądź osiągnie wynik 21 kończy grę. Po zakończeniu rozgrywki przez dwóch graczy, sprawdzane jest, który z graczy wygrał, tj. znalazł się najbliżej liczby 21 (lub wyciągnął dwa asy), nie przekraczając tej wartości. W przypadku przekroczenia wartości przez dwóch graczy, wygrywa gracz, który jest bliżej wartości. O wyniku końcowym powiadamia stosowny komunikat

\* Dla uproszczenia, wyciąganie kolejnych kart nie wpływa na prawdopodobieństwo wyciągnięcia innych kart.

### 3. Arena

Gracz wciela się w rolę gladiatorka, którego głównym celem gracza jest pokonanie finałowego przeciwnika. Podstawowe statystyki gracza to:

- aktualne punkty życia (akthp),
- maksymalne punkty życia (maxhp),
- złoto (zloty),
- poziom (poziom),
- zdobyte doświadczenie (exp),
- liczba mikstur leczniczych (napoje).

Utworzyć diagram, który pozwoli graczowi na branie udziału w walkach.

Implementacja:

- po wejściu do pętli głównej, gracz posiada możliwość wykonania jednej z poniższych czynności:
  - przespanie się w tawernie (odnowienie akthp do poziomu maxhp za wylosowaną liczbę złota),
  - kupienie dodatkowej mikstury leczniczej za określoną ilość złota (zmniejszenie złota oraz zwiększenie zmiennej napoje)
  - wybranie się na walkę z przeciwnikiem prostym – w oddzielnej pętli losowane jest życie przeciwnika, jego obrażenia oraz obrażenia gracza. Dla uproszczenia, walkę jako pierwszy rozpoczyna gracz (można dodać losowanie kolejności) – życie przeciwnika zmniejszane jest o obrażenia gracza, w przypadku wartości równej, bądź mniejszej od zera wygrywa gracz. W przypadku, gdy życie przeciwnika jest większe od zera, przeciwnik zadaje wylosowane obrażenia graczowi. Po pokonaniu wroga gracz otrzymuje ustaloną ilość złota oraz doświadczenia. W przypadku uzyskania doświadczenia powyżej wybranego poziomu, następuje wyzerowanie zmiennej exp, zwiększenie zmiennej poziom oraz zwiększenie życia bohatera.
  - wybranie się na walkę z przeciwnikiem trudnym (patrz przeciwnik prosty – różnica parametrów)



Fundusze Europejskie  
Wiedza Edukacja Rozwój



Rzeczpospolita  
Polska



Unia Europejska  
Europejski Fundusz Społeczny

- walka z finałowym przeciwnikiem (patrz przeciwnik prosty) – w walce może wziąć udział jedynie gracz na poziomie trzecim lub wyższym

Dodatkowe informacje:

- przed rozpoczęciem walki można wylosować kolejność (gracz, czy przeciwnik),
- do pętli walki można dodać wybór pomiędzy dwoma czynnościami: atakiem wroga lub użyciem mikstury leczniczej (o ile ich liczba jest wystarczająca),
- grę można rozszerzyć o dodatkowe elementy (np. kupno miecza, nieudany atak)

**Kącik autora**

Ten niepozorny program może się okazać większy, niż można by przypuszczać!



**Fundusze Europejskie**  
Wiedza Edukacja Rozwój



**Rzeczpospolita Polska**

**Unia Europejska**  
Europejski Fundusz Społeczny



#### 4. Kalkulator

Stworzyć kalkulator, w którym użytkownik może wybrać jedną z poniższych opcji:

- 0 – wyjście,
- 1 – dodawanie,
- 2 – odejmowanie,
- 3 – mnożenie,
- 4 – dzielenie,
- 5 – potęgowanie (implementacja bez wykorzystania wbudowanej funkcji),

Program w pętli głównej pyta użytkownika o wybór opcji – dla wartości różnej od 0 pobierana jest kolejna liczba (1 – liczba zostanie dodana do aktualnego wyniku, 2 – liczba zostanie odjęta od aktualnego wyniku, 3 – wynik zostanie przemnożony przez liczbę, 4 – wynik zostanie podzielony przez liczbę, 5 – wynik zostanie podniesiony do potęgi określonej przez liczbę).

Przykładowy przebieg programu:

'Aktualna wartosc = 0'

'Podaj operacje (0 - wyjscie, 1 [+], 2 [-], 3 [\*], 4 [/], 5[^]): '

[użytkownik wpisuje 1]

'Podaj liczbe do dodania: '

[użytkownik wpisuje 2] -> wynik = wynik+2

'Aktualna wartosc = 2'

'Podaj operacje (0 - wyjscie, 1 [+], 2 [-], 3 [\*], 4 [/], 5[^]): '

[użytkownik wpisuje 5]

'Podaj wykładnik potęgi: '

[użytkownik wpisuje 8] -> wynik = wynik^8

'Aktualna wartosc = 256'

'Podaj operacje (0 - wyjscie, 1 [+], 2 [-], 3 [\*], 4 [/], 5[^]): '

[użytkownik wpisuje 0]

'Dziękujemy za skorzystanie z naszego kalkulatora'



Fundusze  
Europejskie  
Wiedza Edukacja Rozwój



Rzeczpospolita  
Polska



## ROZDZIAŁ 4. WPROWADZENIE DO TABLIC.

### Cel rozdziału:

Przedstawienie budowy tablicy oraz sposobów wykorzystania tablic w praktyce.

### Zakres tematyczny:

1. Omówienie budowy tablicy.
2. Omówienie podstawowych operacji na tablicach (oraz ich związku z pętlami).
3. Przedstawienie sposobu użycia tablicy na prostym przykładzie.
4. Rozwiązywanie zadań z użyciem pętli.

### Pytania kontrolne:

1. W jaki sposób pobrać od użytkownika liczbę?
2. Z jakich sekcji składa się instrukcja warunkowa?
3. W jakim celu wykorzystywane są pętle?
4. Jak wygląda budowa pętli?
5. Jakie konsekwencje niesie ze sobą pętla nieskończona?

### 1. Idea tablicy

Większość dotychczasowych przykładów potrzebowało jedynie kilku zmiennych do przechowywania bieżącego stanu aplikacji. Czasami zdarza się jednak, że aplikacja lub gra potrzebuje przechowywać tych danych znacznie więcej. Przykładowo, nawet w najprostszego grach strategicznych na ekranie może się poruszać się jednocześnie ponad kilkaset pojazdów, które trzeba narysować i sprawdzić kolizje z innymi obiektami na mapie – przechowywanie tak dużego zbioru danych w oddzielnych zmiennych nie jest zbyt dobrym pomysłem.

#### Kącik autora

Nawet dla 10 pojazdów znajdujących się na mapie, sprawdzenie kolizji pomiędzy nimi (na zasadzie „każdy z każdym”) bez użycia tablic jest naprawdę dużym wyzwaniem i bardzo łatwo o popełnienie błędu przy tworzeniu instrukcji warunkowych (czy pamiętaś z matematyki, ile warunków musiałoby zostać sprawdzonych dla 10 pojazdów?) Na szczęście od tego momentu będziesz mógł bez problemu wykorzystać do tego zadania tablice!

W dużym uproszczeniu, tablica stanowi szereg powiązanych zmiennych ułożonych w sposób ciągły w pamięci komputera. Tablice pozwalają na przechowywanie i przetwarzanie dużych zbiorów danych w relatywnie prosty sposób – dostęp do każdego elementu możliwy jest poprzez podanie numeru (indeksu) konkretnej komórki. W większości języków programowania (również w Pythonie) numerowanie komórek rozpoczyna się od indeksu zerowego (choć istnieją języki, w których pierwszy element ma indeks równy jeden).

Na rys. 5.1 przedstawiono schemat poglądowy niewielkiej, pięcioelementowej tablicy liczb typu całkowitego.



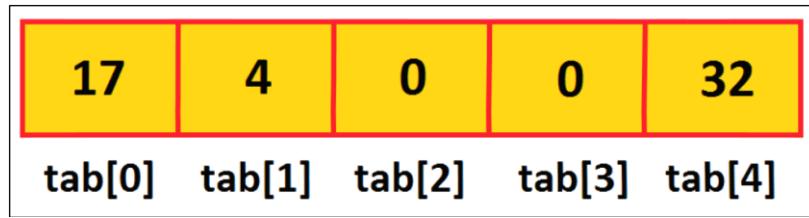
Fundusze Europejskie  
Wiedza Edukacja Rozwój



Rzeczpospolita  
Polska



Unia Europejska  
Europejski Fundusz Społeczny



Rys. 5.1. Tablica o nazwie tab, która posiada 5 elementów o wartościach: 17,4,0,0,32

Nieco niżej na rys. 5.2 przedstawiono inicjalizację oraz najprostszy rodzaj dostępu do tablicy jednowymiarowej, tj. poprzez podanie numeru konkretnej komórki w formie liczbowej. Na końcu wyświetlono stan tablicy po modyfikacjach. W pojedynczej komórce tablicy można przechowywać zarówno wszystkie typy proste (liczby, tekst, typ logiczny), jak i inne tablice! (właściwość ta zostanie wykorzystana w rozdziałach dotyczących budowy tablic wielowymiarowych). Warto zwrócić uwagę na fakt, iż w związku z faktem numeracji komórek od zera, ostatni element jest dostępny pod indeksem równym 4 (a nie 5, jak można by było przypuszczać po rozmiarze tablicy).

1	<code>tab = [17, 4, 0, 0, 32]</code>
2	
3	<code>tab[0] += 13</code>
4	<code>tab[4] = 10</code>
5	
6	<code>print(tab)</code>

Rys. 5.2. Deklaracja tablicy od nazwie tab, modyfikacja pierwszego i ostatniego elementu oraz wypisanie stanu tablicy w konsoli

Najbardziej praktycznym sposobem dostępu do tablic jest skorzystanie z pętli i dodatkowej zmiennej pełniącej funkcję licznika – przykłady użycia zostaną przedstawione w dalszej części rozdziału.

## 2. Reprezentacja na diagramie NS

Jako, że tworzenie oraz dostęp do elementów tablicy zawiera się w ramach podstawowych operacji na danych, diagram NS nie przewiduje nowego symbolu graficznego – wszystkie operacje na tablicach wykonywane są w ramach pustego prostokąta, zgodnie z rys. 5.3.



Fundusze  
Europejskie  
Wiedza Edukacja Rozwój



Rzeczpospolita  
Polska



```
tab = [17, 4, 0, 0, 32]
tab[0] += 13
tab[4] = 10
tab
```

Rys. 5.3. Deklaracja tablicy od nazwie *tab*, modyfikacja pierwszego i ostatniego elementu oraz wypisanie zawartości tablicy

#### Kącik autora

Mimo stosunkowo prostej budowy, tablice w połączeniu z pętlami potrafią być momentami sporym problemem, w szczególności dla osób, które nie miały wcześniej styczności z programowaniem. W momencie, gdy analiza kodu staje się coraz trudniejsza, warto skorzystać z trybu pracy krokowej, który pozwoli obserwację zmiany stanu tablicy.

### 3. Operacje na tablicach

W związku z faktem, iż w języku Python tablice są reprezentowane przy użyciu list (tj. struktur, które mogą dynamicznie zmieniać swój rozmiar), korzystanie z tablic wymaga omówienia sytuacji, które mogą się wydawać początkowo nieintuicyjne.

#### 1. Deklaracja tablicy o określonym rozmiarze

Najprostszym sposobem utworzenia tablicy o określonym rozmiarze jest zastosowanie techniki wielokrotnego powielenia wybranej wartości (najczęściej będzie to zero). Operacja ta przyjmuje następującą postać:

```
tablica = [0]*30
print(tablica)
```

Efektem wykonania powyższego kodu będzie utworzenie trzydziestoelementowej tablicy wypełnionej zerami. W zadaniach zazwyczaj pojawia się zapis mówiący o tym, co określa rozmiar tablicy (najczęściej występujący pod nazwą *n*). W większości przypadków wartość ta jest pobrana wcześniej od użytkownika i może zostać bez problemu użyta do utworzenia tablicy:

```
n = int(input('Podaj rozmiar tablicy: '))
tablica = [0]*n
print(tablica)
```

Wcześniejsza inicjalizacja rozmiaru tablicy nie jest potrzebna w przypadku korzystania z diagramów NS.



Fundusze  
Europejskie  
Wiedza Edukacja Rozwój



Rzeczpospolita  
Polska



Unia Europejska  
Europejski Fundusz Społeczny

## 2. Pobranie od użytkownika n wartości i zapisanie ich do tablicy

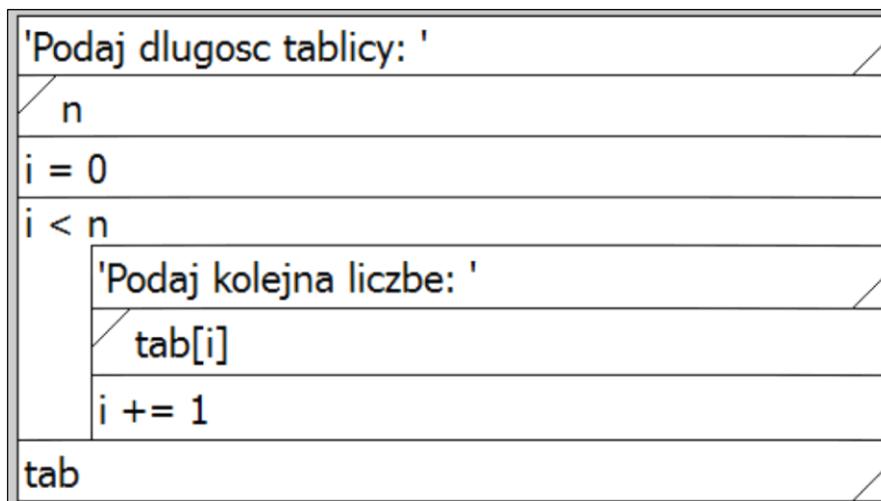
Stosunkowo często występują sytuacje, w których wykonanie programu rozpoczyna się od pobrania od użytkownika długości tablicy, a następnie pobrania n wartości i wpisania ich do tablicy.

```
n = int(input('Podaj dlugosc tablicy: '))
tab = [0]*n

i = 0
while i < n:
    tab[i] = float(input('Podaj kolejna liczbe: '))
    i += 1

print(tab)
```

Do realizacji zadania została użyta pętli `while` z licznikiem w postaci zmiennej `i`. Pętla wykona się dokładnie n razy, począwszy od indeksu 0, a skończywszy na `n-1`. Rezultatem działania kodu będzie tablica wypełniona liczbami zmiennoprzecinkowymi. Rys. 5.4 przedstawia reprezentację na diagramie.



Rys. 5.4. Pobranie do tablicy n wartości od użytkownika

## 3. Wyświetlenie wybranych elementów tablicy

O ile wyświetlenie wszystkich elementów tablicy jest stosunkowo proste wykorzystując funkcję `print`, o tyle wyświetlenie wybranych elementów tablicy jest już nieco bardziej skomplikowane. Przykładowo, zadanie wymaga wyświetlenia w konsoli wszystkich nieparzystych elementów tablicy. Zadanie można to zrealizować przynajmniej na dwa różne sposoby.

Pierwszy z nich opiera się na wykorzystaniu przedstawionej wcześniej pętli `while` ze zmienną sterującą. Aby przejrzeć wszystkie elementy tablicy należy rozpocząć liczenie od indeksu zerowego, aż do zdefiniowanej wcześniej długości tablicy, każdorazowo sprawdzając, czy aktualny element tablicy spełnia warunek nieparzystości. W kodzie pominięto moment pobrania danych.



Fundusze Europejskie  
Wiedza Edukacja Rozwój



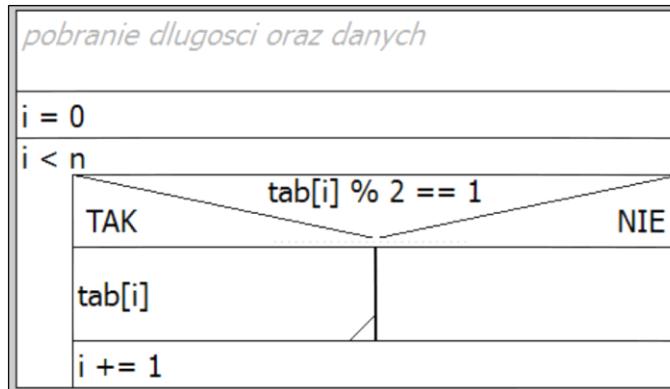
Rzeczpospolita  
Polska



Unia Europejska  
Europejski Fundusz Społeczny

```
# deklaracja tablicy oraz pobranie danych od użytkownika  
  
i = 0  
while i < n:  
    if tab[i] % 2 == 1:  
        print(tab[i])  
  
    i += 1
```

Analogicznie wygląda reprezentacja w postaci diagramu NS (rys. 5.5)



Rys. 5.5. Wyświetlenie wszystkich liczb nieparzystych znajdujących się w tablicy

Drugą metodą wyświetlenia, która nie posiada reprezentacji na diagramie NS jest wykorzystanie pętli `for-each` do przejrzenia wszystkich elementów znajdujących się w tablicy. Użycie pętli `for-each` sprawia, że można zupełnie pominąć użycie zmiennej sterującej `i` – przykład zamieszczono poniżej:

```
# deklaracja tablicy oraz pobranie danych od użytkownika  
  
for element in tab:  
    if element % 2 == 1:  
        print(element)
```

Jak widać, pętla `for-each` zdecydowanie upraszcza dostęp do wartości przechowywanych w tablicy i minimalizuje ryzyko popełnienia błędu (dzięki usunięciu zmiennej sterującej).



Fundusze Europejskie  
Wiedza Edukacja Rozwój



Rzeczpospolita  
Polska

Unia Europejska  
Europejski Fundusz Społeczny



### Zadania do samodzielnej realizacji [klasyczne]

1. Pobrać od użytkownika n wartości umieszczając je w tablicy, a następnie znaleźć i wyświetlić dwa największe elementy.
2. Wczytać od użytkownika n wartości. Po uzupełnieniu tablicy pobrać dwie wartości określające obszar poszukiwań – sprawdzić i wypisać ile elementów tablicy mieści się w wyznaczonym przedziale.
3. Pobierać od użytkownika wartości do momentu wpisania wartości zero. Sprawdzić, czy tablica jest symetryczna względem jej środka. Uwzględnić zarówno tablice o parzystej, jak i nieparzystej liczbie elementów.
4. Pobierać od użytkownika wartości do momentu wpisania wartości zero. Znaleźć i wypisać wszystkie elementy, które pojawiły się w tablicy więcej niż dwa razy.
5. Wczytać od użytkownika n wartości (ponowić wczytanie, gdy  $n < 4$ ). Znaleźć w tablicy dwa miejsca, dla których suma trzech elementów jest największa (jako wynik wypisać dwa indeksy oraz dwie największe sumy).

### Zadania do samodzielnej realizacji [tekstowe]

1. Loteria

Wypełnić tablicę sześcioma losowymi wartościami z przedziału 1 – 40. Następnie pobrać od użytkownika również sześć liczb z tego samego przedziału (w przypadku podania liczby spoza zakresu lub liczby już podanej, ponownie wczytać wartość).

Sprawdzić, czy użytkownik wygrał w loterii – wyświetlić odpowiedni komunikat w zależności od liczby takich samych wartości pomiędzy tablicami:

- 3 lub mniej takich samych wartości – brak wygranej,
- 4 – wygrana 50 zł,
- 5 – wygrana 400 zł,
- 6 – wygrana 2000 zł.

2. Zaktualizować grę planszową z rozdziału trzeciego tak, aby plansza była przechowywana w tablicy.



Fundusze Europejskie  
Wiedza Edukacja Rozwój



Rzeczpospolita  
Polska



3. Zaprojektować i zaimplementować grę „Memo”, w której:

1. Utworzyć tablicę, wypełnić ją 8 parami wartości:

[1,1,2,2,3,3,4,4,5,5,6,6,7,7,8,8], a następnie przetasować.

2. Wejść do głównej pętli gry, gdzie poprosić gracza o podanie dwóch indeksów tablicy które chce sprawdzić. Wyświetlić graczowi informacje o wartościach znajdujących się pod wybranymi indeksami. W przypadku trafienia takich samych wartości, zwiększyć licznik znalezionych par, jednocześnie zerując w tablicy odkryte pola (żeby nie można było ich ponownie odkryć).

3. Gra kończy się w momencie, gdy gracz odkryje wszystkie pola.



**Fundusze  
Europejskie**  
Wiedza Edukacja Rozwój



**Rzeczpospolita  
Polska**

**Unia Europejska**  
Europejski Fundusz Społeczny



## **ROZDZIAŁ 5. POWTÓRZENIE MATERIAŁU – ROZWIĄZYwanie ZADAŃ, ANALIZA WYBRANYCH ROZWIĄZAŃ**

### **Cel rozdziału:**

Ugruntowanie zdobytych umiejętności w zakresie przeprowadzania operacji na danych liczbowych.

### **Zakres tematyczny:**

1. Realizacja algorytmów związanych z użyciem instrukcji warunkowych i pętli
2. Rozwiązywanie zadań z zastosowaniem tablic
3. Analiza zamieszczonych diagramów NS
4. Przeniesienie rozwiązania w formie graficznej do postaci implementacyjnej.

### **Pytania kontrolne:**

1. W jaki sposób pobrać od użytkownika liczbę?
2. Z jakich sekcji składa się instrukcja warunkowa?
3. W jakim celu wykorzystywane są pętle?
4. Jak wygląda budowa pętli?
5. Jak jest zbudowana tablica i jak się dostać do jej elementów?
6. Jaki związek mają ze sobą pętle i tablice?
7. Czy da się korzystać z tablicy bez tworzenia pętli?

#### **Kącik autora**

Ze względu na ograniczony czas na kolokwium, zazwyczaj unika się umieszczenia na nim zadań rozległych i rozbudowanych fabularnie – priorytetem jest sprawdzenie umiejętności studenta w rozwiązywaniu konkretnych problemów.

### **Zadania do samodzielnej realizacji**

1. W pętli wczytywać od użytkownika kolejne wartości. Podanie liczby zero powoduje wejście lub wyjście z trybu liczenia średniej (tylko wtedy liczby będą uwzględnione podczas liczenia średniej) – początkowo tryb jest wyłączony. Podanie liczby ujemnej powoduje wyjście z pętli (tylko w momencie, gdy tryb liczenia średniej jest wyłączony) i wyświetlenie obliczonej średniej. Dwa przykładowe uruchomienia:

4 5 7 0 3 1 2 0 2 1 7 0 6 0 8 2 8 9 -1 ->  $(3+1+2+6) / 4 \rightarrow 3$

2 0 2 1 3 0 7 -1 ->  $(2+1+3) / 3 \rightarrow 2$

2. Pobrać od użytkownika rozmiar tablicy n, a następnie wczytać n podanych przez użytkownika wartości do tablicy. Obliczyć i wyświetlić sumę oraz średnią wszystkich elementów większych od ostatniego. Przygotować diagram NS oraz zaimplementować algorytm w języku Python.



**Fundusze Europejskie**  
Wiedza Edukacja Rozwój



**Rzeczpospolita Polska**



**Unia Europejska**  
Europejski Fundusz Społeczny

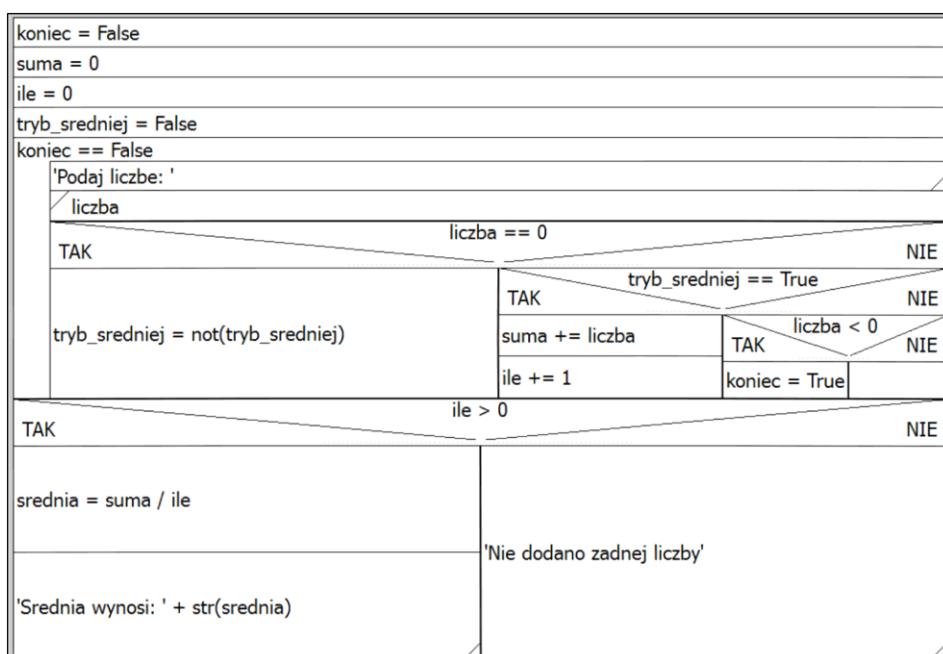
3. Pobrać od użytkownika rozmiar tablicy n, a następnie wczytać n podanych przez użytkownika liczb całkowitych do tablicy. Odnaleźć liczby, których suma cyfr jest większa od ostatniego elementu podanego przez użytkownika. Znalezione liczby umieścić w drugiej tablicy. Przed wyjściem z programu wyświetlić ich ilość.
4. Pobierać od użytkownika liczby do momentu wpisania liczby zero, bądź pobrania dziesięciu wartości. Po zakończonym wczytywaniu ustawić liczby tak, aby liczby parzyste znalazły się po jednej stronie tablicy, a nieparzyste po drugiej. Wyświetlić zawartość tablicy.
5. Pobrać od użytkownika n wartości i zapisać je do tablicy. Po pobraniu wszystkich elementów, usunąć z tablicy wszystkie wielokrotności liczby 5 – proces usunięcia polega na przesunięciu wszystkich elementów tablicy znajdujących się po prawej stronie od elementu usuwanego o jedną pozycję w lewo. Wyświetlić zmodyfikowaną postać tablicy.

**Kącik autora**

Zachęcam do samodzielnego zastanowienia się nad rozwiązaniem problemu i dopiero w ostateczności wspomożenia się zamieszczonymi niżej diagramami.

**Rozwiązania do zadań w postaci diagramów NS**

1. Zadanie, do którego rozwiązania nie jest potrzebne użycie tablic (choć mogą to lekko sugerować dwa przykładowe uruchomienia ukazane w treści zadania). Głównym problemem zadania jest opracowanie zależności pomiędzy wprowadzoną wartością, a aktualnym stanem trybu liczenia średniej. Jedno z przykładowych rozwiązań tego zadania zostało przedstawione na rys. 5.1.



Rys. 5.1. Diagram referencyjny zadania nr 1



**Fundusze Europejskie**  
Wiedza Edukacja Rozwój

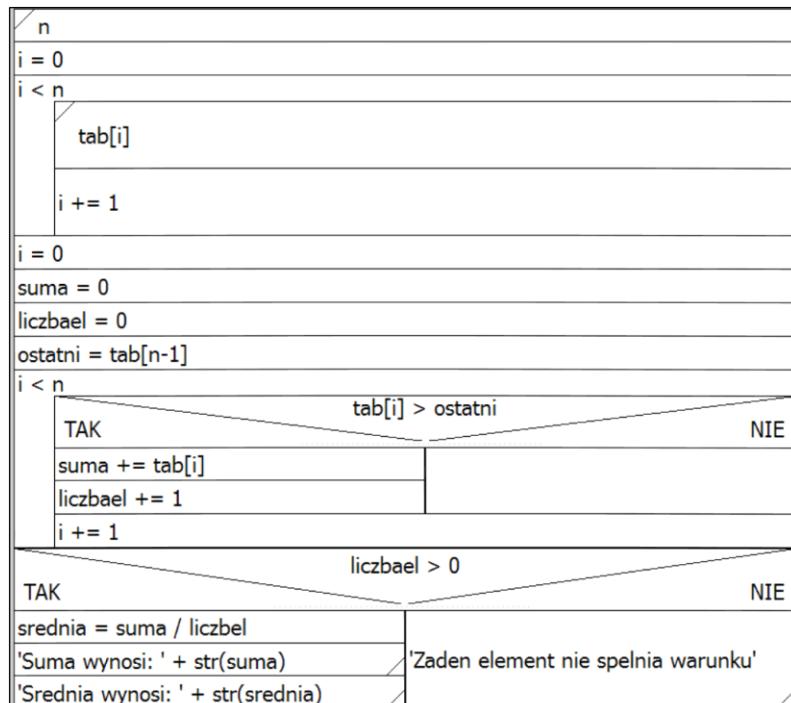


**Rzeczypospolita  
Polska**

**Unia Europejska**  
Europejski Fundusz Społeczny



2. Klasyczne zadanie, w którym zawarto: pobranie danych od użytkownika oraz obliczenie sumy i średniej wprowadzonych wartości z dodatkowym obostrzeniem (liczbybrane pod uwagę muszą być większe od ostatniego elementu, stąd obecność zapisu: `tab[i] > ostatni`). Diagram referencyjny umieszczono na rys. 5.2.



Rys. 5.2. Diagram referencyjny zadania nr 2

3. Trzecie zadanie posiada strukturę zbliżoną do zadania drugiego – pobranie danych od użytkownika, a następnie ich przetworzenie w osobnej pętli. Tym razem zadanie łączy ze sobą pracę na tablicach z umiejętnością wyciągnięcia cyfr składających się na liczbę – efektem tego rozwiązania jest wystąpienie zagnieżdżonej pętli `while` dokonującej rozbicia liczby na składowe. Diagram referencyjny umieszczono na rys. 5.3.

Na diagramie NS nie występuje problem deklaracji rozmiaru drugiej tablicy – podczas implementacji w języku Python najbezpieczniej jest nadać drugiej tablicy również rozmiar początkowy równy `n`.



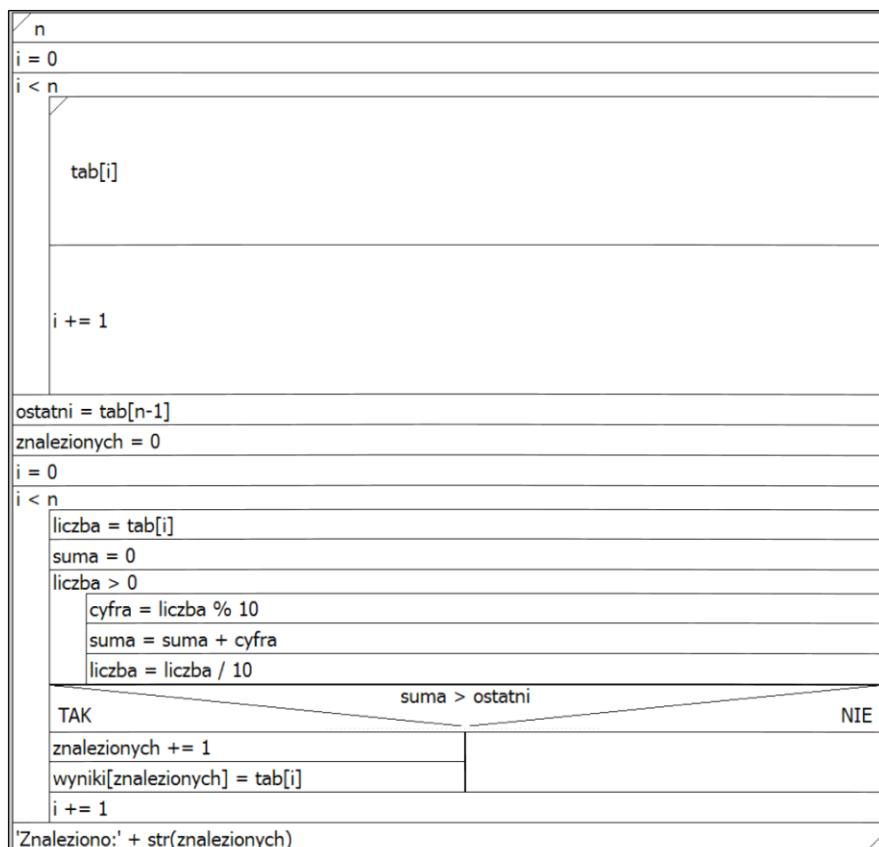
Fundusze Europejskie  
Wiedza Edukacja Rozwój



Rzeczpospolita  
Polska

Unia Europejska  
Europejski Fundusz Społeczny





Rys. 5.3. Diagram referencyjny zadania nr 3

- W zadaniu czwartym (rys. 5.4) należy zastosować nieco inne podejście przy wczytywaniu danych w związku z faktem, iż nieznana jest ilość liczb, które zostaną wprowadzone przez użytkownika. Zastosowana konstrukcja pozwala na obejście tego problemu.

Po prawidłowym wczytaniu danych pozostaje już tylko ustawić elementy w tablicy zgodnie z treścią zadania – elementy parzyste mają się znaleźć po lewej stronie, nieparzyste po prawej. Na wstępnie należy zwrócić uwagę na fakt, iż treść zadania nie preczyzuje, czy liczby po obu stronach mają być w jakikolwiek sposób posortowane, stąd nie muszą być uporządkowane rosnąco.

Zaproponowana idea rozwiązania polega na wyznaczeniu dwóch indeksów przyjmujących odpowiednio nazwy: lewy ora prawy. Lewy indeks wskazuje na elementy parzyste (lewa strona tablicy), podczas gdy prawy indeks wskazuje na elementy nieparzyste (prawa strona). W pętli pobierane są kolejno elementy poczynając od lewej strony tablicy i sprawdzana jest ich parzystość – jeżeli element jest już parzysty można przejść dalej. W przypadku wystąpienia elementu nieparzystego należy dokonać zamiany z elementem z prawej strony tablicy. Jako, że nie wiadomo, czy element po prawej stronie jest parzysty, przy zamianie nie jest zwiększany lewy indeks – pozwoli to na sprawdzenie parzystości zamienionego elementu w następnym cyklu pętli. Pętla kończy działanie, gdy lewy indeks minie się z indeksem prawym.



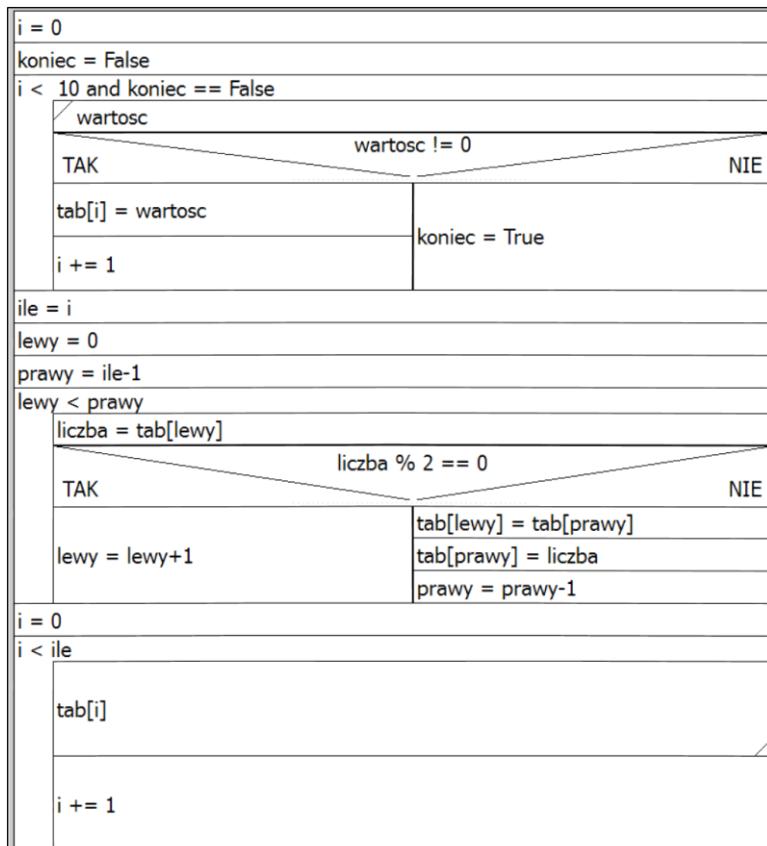
Fundusze Europejskie  
Wiedza Edukacja Rozwój



Rzeczypospolita  
Polska

Unia Europejska  
Europejski Fundusz Społeczny





Rys. 5.4. Diagram referencyjny zadania nr 4

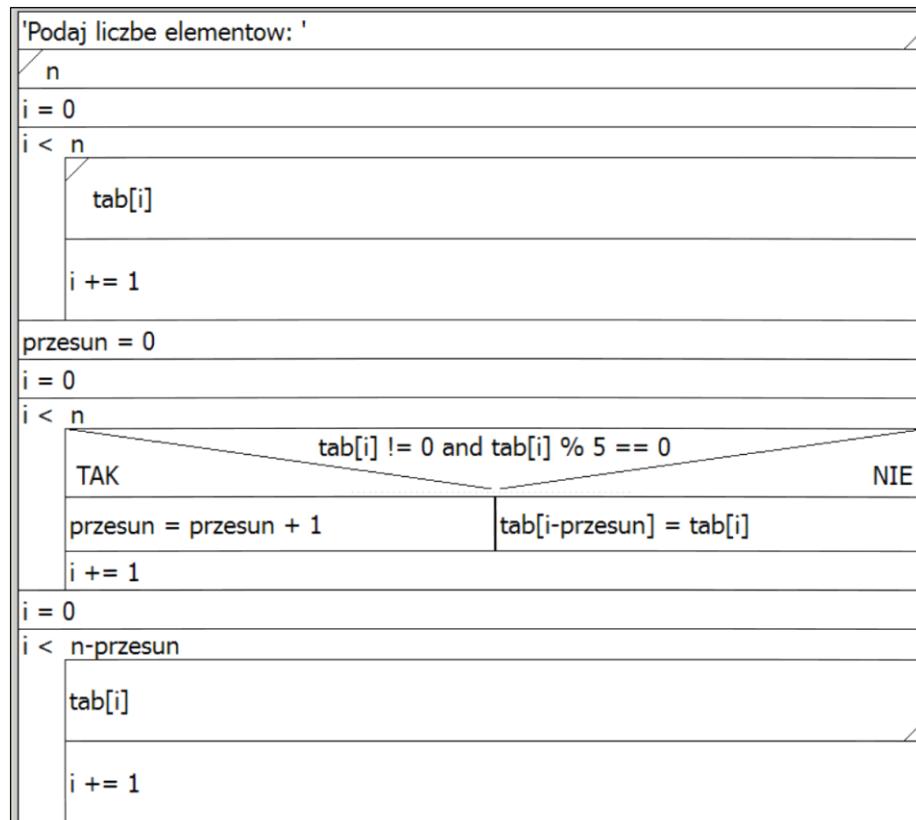
Przykład ten warto przeanalizować na spokojnie w trybie pracy krokowej lub na diagramie NS (z użyciem przykładowych danych) obserwując zmianę wartości zmiennych `lewy` oraz `prawy`.

5. Diagram (rys. 5.5) standardowo rozpoczyna się od pobrania wartości od użytkownika. Zadaniem programisty jest odnalezienie wszystkich wielokrotności liczby 5 i usunięciu ich z tablicy tak, aby zachowana była ciągłość elementów wprowadzonych. Oznacza to, że za każdym razem, gdy odnaleziona zostanie wielokrotność liczby 5, niezbędne jest przesunięcie wszystkich elementów znajdujących się po prawej stronie o jedną pozycję w lewo. Zadanie to można zrealizować w oparciu o dodatkową, zagnieżdżoną pętlę, która przesuwałaby wszystkie elementy od aktualnego indeksu do końca tablicy o jedną pozycję w lewo, lecz takie podejście wykazuje większą złożoność obliczeniową (większe zużycie procesora) względem rozwiązania zaproponowanego. W przedstawionym na rys. 5.5 rozwiążaniu wykorzystano dodatkową zmienną, która zapamiętuje o ile komórek w lewo należy przesunąć kolejne elementy tablicy.

Początkowo wartość zmiennej `przesun` wynosi 0 – jeżeli nie napotkano jeszcze żadnego elementu modulo 5, przesuwanie elementów tablicy nie jest potrzebne. Oznacza to, że prawa część instrukcji warunkowej przyjmie postać:  $tab[i - 0] = tab[i]$ , co w praktyce oznacza brak zmian w zawartości tablicy.



Wystąpienie chociażby jednego elementu modulo 5 zwiększa wartość zmiennej i od tego momentu każdy kolejny element, który nie spełnia warunku jest przypisywany o jedną pozycję bliżej początku tablicy, co odpowiada wymaganiom zadania.



Rys. 5.5. Diagram referencyjny zadania nr 5

Bardziej zaawansowane przykłady z użyciem tablic zostaną omówione w ramach oddzielnego laboratorium poświęconemu użyciu tablic wielowymiarowych oraz pętli wielokrotnie zagnieżdżonych.



Fundusze  
Europejskie  
Wiedza Edukacja Rozwój



Rzeczpospolita  
Polska

Unia Europejska  
Europejski Fundusz Społeczny



## **ROZDZIAŁ 6. ROZWIĄZYWANIE ZADAŃ DO SAMODZIELNEJ REALIZACJI**

### **Cel rozdziału:**

Sprawdzenie umiejętności studenta w zakresie:

- posługiwania się zmiennymi,
- interakcji z użytkownikiem (pobranie wartości, wypisanie wartości),
- konstruowania złożonych instrukcji warunkowych,
- kontrolowania przebiegu pętli,
- operowania na tablicach jednowymiarowych,
- szeroko pojętej algorytmiki.
- interakcji z użytkownikiem (pobranie wartości, wypisanie wartości)

### **Zakres tematyczny:**

1. Powtórzenie dotychczas zdobytej wiedzy oraz sprawdzenie swoich umiejętności programowania podczas rozwiązywania zadań.

### **Pytania kontrolne:**

Brak

#### **1. Zestaw nr 1**

1. Wczytać cztery liczby dodatnie. Podzielić trzy pierwsze liczby przez liczbę ostatnią, każdorazowo wyświetlając wynik.
2. Wczytać n liczb, a następnie wyświetlić sumę ich kwadratów.
3. Wczytywać liczby do momentu wystąpienia liczby będącej wielokrotnością liczby 10. Wyświetlić najmniejszy element oraz średnią wczytanych liczb.
4. Wczytać tablicę n liczb. Znaleźć trzy kolejne liczby w tablicy, których suma jest największa. Wyświetlić numer indeksu początkowego znalezionych liczb.



**Fundusze Europejskie**  
Wiedza Edukacja Rozwój



**Rzeczpospolita Polska**



**Unia Europejska**  
Europejski Fundusz Społeczny

## 2. Zestaw nr 2

1. Wczytać n liczb, a następnie wyświetlić największą i najmniejszą z nich.
2. Wczytaj liczbę oraz wykładnik potęgi (liczby dodatnie). Wyświetlić wynik podniesienia liczby do danej potęgi (bez wykorzystania wbudowanych funkcji).
3. Pobrać od użytkownika ilość liczb do wczytania (w przypadku podania wartości mniejszej od dwóch lub nieparzystej, wyjść z programu). W pętli wczytać od użytkownika podaną ilość liczb. W wyniku wyświetlić średnią dwóch liczb znajdujących się pośrodku wczytanego zbioru. Przykład:

2,5,3,4,1,9

wynik (średnia): 3,5

1,5,7,7

wynik (średnia): 6

4. Pobrać od użytkownika n liczb całkowitych od, przyjmujących wartości od 1 do 100 (w przypadku niespełnienia warunku pobrać liczbę ponownie). Sprawdzić, czy w podanej tablicy istnieje choć jedna para takich samych liczb.

## 3. Zestaw nr 3

1. Wczytywać od użytkownika liczby do momentu podania liczby zero. Wyświetlić sumę liczb mniejszych od pierwszej podanej liczby.
2. Wczytać od użytkownika dwie wartości dodatnie odpowiadające podstawie i wykładnikowi potęgi. Następnie pobrać wartość (również dodatnią) ogranicznika. Sprawdzić poprawność wprowadzonych liczb (w przypadku niespełnionego warunku wyświetlić odpowiedni komunikat i wyjść z programu). Wyświetlić liczbę podniesioną do potęgi, tak aby wynik nie był większy od ograniczenia.

Przykładowe działanie programu:

2,8,100 # 2<sup>8</sup> ograniczone do 100 -> najbliższa wartość przed ogranicznikiem: 2<sup>5</sup>

wynik: 64

2,5,10 # 2<sup>5</sup> ograniczone do 10 -> najbliższa wartość przed ogranicznikiem: 2<sup>3</sup>

wynik: 8

3. Wczytywać elementy od użytkownika do momentu podania liczby spoza przedziału 10 – 500 (wartość spoza przedziału nie jest brana pod uwagę w dalszej części zadania). Znaleźć i wyświetlić dwa najmniejsze elementy.



Fundusze Europejskie  
Wiedza Edukacja Rozwój



Rzeczpospolita  
Polska



Unia Europejska  
Europejski Fundusz Społeczny

4. Pobrać od użytkownika n liczb rzeczywistych, a następnie znaleźć najdłuższy, niemalejący ciąg liczb. W wyniku wyświetlić indeks początkowy ciągu i jego długość.



**Fundusze  
Europejskie**  
Wiedza Edukacja Rozwój



**Rzeczpospolita  
Polska**

**Unia Europejska**  
Europejski Fundusz Społeczny



## **LABORATORIUM 2. PROCEDURY I FUNKCJE, REKURENCJA.**

### **Cel laboratorium:**

Przedstawienie budowy oraz zasady użycia procedur oraz funkcji.

### **Zakres tematyczny zajęć:**

1. Procedury i funkcje – tworzenie, wywoływanie i przekazywanie wartości
  - a. Zastosowania funkcji.
  - b. Różnica między funkcją, a procedurą.
  - c. Przekazywanie wartości do procedury/funkcji. Zwracanie wartości przez funkcję.
  - d. Omówienie rozbicia przykładowego kodu na mniejsze składowe.
2. Rekurencja
  - a. Pojęcie rekurencji
  - b. Omówienie przykładu wykorzystującego rekurencję.

### **Pytania kontrolne:**

Brak

## **ROZDZIAŁ 7. PROCEDURY I FUNKCJE – TWORZENIE, WYWOŁYWANIE, PRZEKAZYWANIE I ZWRACANIE WARTOŚCI.**

### **Cel rozdziału:**

Celem rozdziału jest zapoznanie czytelnika z pojęciem procedury oraz funkcji – w jaki sposób z nich korzystać oraz jakie mają zastosowania.

### **Zakres tematyczny:**

1. Zastosowania funkcji.
2. Różnica między funkcją, a procedurą.
3. Przekazywanie wartości do procedury/funkcji. Zwracanie wartości przez funkcję.
4. Omówienie rozbicia przykładowego kodu na mniejsze składowe.

### **Pytania kontrolne:**

1. Jakie wyróżniamy typy zmiennych?

### **1. Zastosowanie funkcji**

Wraz ze wzrostem złożoności tworzonych programów, niezbędne staje się korzystanie z technik pozwalających na łatwiejsze zarządzanie kodem źródłowym – jedną z takich technik jest zastosowanie procedur i funkcji. W dużym skrócie, procedury i funkcje są wydzielonymi fragmentami kodu.

Zalety korzystania z procedur i funkcji:

- Funkcje pozwalają na usprawnienie zarządzania kodem programu, poprzez wydzielenie fragmentów kodu odpowiedzialnych za wykonanie konkretnego zadania – nawet w przypadku, gdy utworzona procedura lub funkcja wywoływana będzie w kodzie tylko jednorazowo. W ten sposób unika się również potrzeby tworzenia komentarzy – dobrze wydzielony i nazwany w postaci funkcji kod powinien jednoznacznie opisywać swoje działanie (w idealnym przypadku, kod programu powinien być zawsze rozbity na funkcje, których długość nie powinna przekraczać kilkudziesięciu/kilkunastu linii kodu).
- Fragmenty kodu wykorzystywane w wielu miejscach mogą zostać przekształcone na funkcje, co pozwoli na redukcję liczbę kodu w programie i pozwoli zminimalizować ryzyko wystąpienia błędów w programie (w przypadku błędu w algorytmie, wystarczy poprawić kod źródłowy tylko w jednym miejscu, a zmiana zostanie uwzględniona w całym programie).
- Raz napisany kod może zostać przekształcony do postaci biblioteki, co pozwoli na wykorzystanie napisanych funkcji w kolejnych programach.



**Fundusze Europejskie**  
Wiedza Edukacja Rozwój



**Rzeczpospolita Polska**



**Unia Europejska**  
Europejski Fundusz Społeczny

- Procedury i funkcje pozwalają na uniezależnienie fragmentów kodu od stanu pozostałych zmiennych, co sprzyja tworzeniu kodu bardziej uniwersalnego (patrz przekształcenie kodu do postaci biblioteki).
- Każda procedura i funkcja powinna wykorzystywać jedynie wartości przekazane za pośrednictwem parametrów (wyjątkiem są zmienne globalne), co oznacza, że zmniejsza się ryzyko przypadkowej zmiany wartości zmiennych niezwiązanych bezpośrednio z funkcją (znacznie łatwiej się zarządza kodem wiedząc, że funkcja może zmienić wartość jedynie trzech zmiennych, a nie stu, czy tysiąca). Zaleta ta jest widoczna wraz ze wzrostem rozmiaru projektu.

Nawet podczas pisania aplikacji konsolowych, praktycznie wszystkie operacje interakcji z użytkownikiem (pobierania tekstu z klawiatury oraz wyświetlanie tekstu na ekranie) są realizowane poprzez użycie napisanych wcześniej funkcji (odpowiednio `input` oraz `print`).

## 2. Różnica między procedurą i funkcją

Można zauważyć, że w większości języków programowania nie istnieje rozgraniczenie pomiędzy nazwą „procedura”, a „funkcja” – wyodrębnione sekcje kodu przyjmują praktycznie zawsze nazwę funkcji. W kilku językach (m.in. w języku Pascal) zdecydowano się jednak na zachowanie podziału bliższego definicji matematycznej.

Z punktu widzenia algorytmiki:

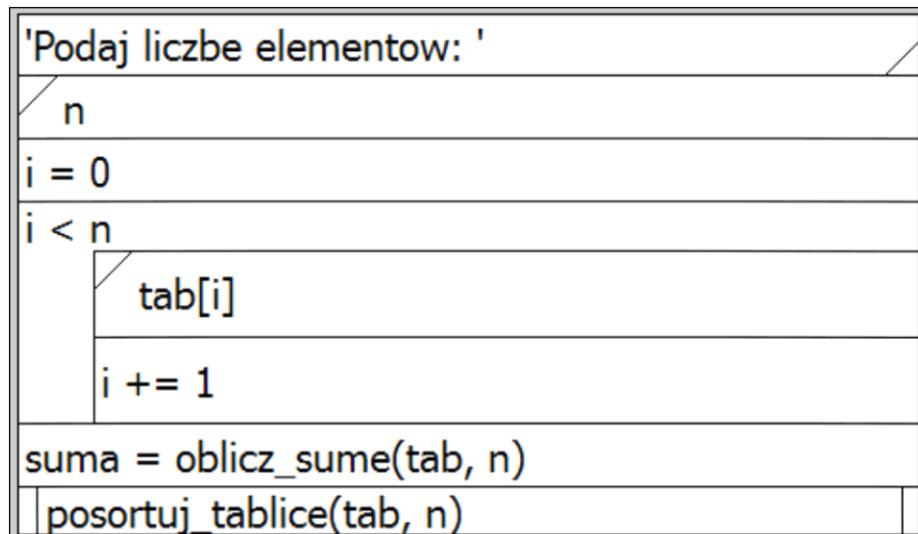
- procedurą nazywamy określona sekwencję działań,
- funkcją nazywamy przekształcenie zbioru wejściowego na wyjściowy w oparciu o określony schemat działania (definicja typowo matematyczna).

Z punktu widzenia języka Python:

- procedurą jest funkcją, która nie zwraca wartości,
- funkcja jest sekwencją działań, która może się zakończyć zwróceniem określonej wartości.

Diagram przedstawiony na rys. 7.1 przedstawia tą subtelną różnicę. Zasadniczo, diagramy NS również stosują rozgraniczenie pomiędzy procedurą, a funkcją – stosowany jest nawet specjalny typ bloczka dla wywołania procedury (rys. 7.1).





Rys. 7.1. Wywołanie procedury i funkcji na jednym diagramie

### 3. Funkcje w języku Python

W języku Python funkcje definiowane są za pomocą słowa kluczowego `def`, po którym następuje: podanie nazwy funkcji, nawiasów okrągłych, opcjonalnych parametrów – całość wieńczy symbol dwukropka. O zasięgu ciała funkcji decyduje wcięcie (podobnie jak w przypadku instrukcji warunkowych i pętli). W zależności od sposobu użycia funkcja może, ale nie musi zwracać wartości. Implementację dwóch stosunkowo prostych funkcji przedstawiono na rys. 7.2.

```
1  def pomnoz(x, y):
2      wynik = x * y
3      return wynik
4
5  def obsluz_mnozenie():
6      liczba1 = int(input('Podaj pierwsza liczbe: '))
7      liczba2 = int(input('Podaj druga liczbe: '))
8
9      print('Wynik mnozenia = ' + str(pomnoz(liczba1, liczba2)) + '\n')
10
11     wybor = 1
12     while wybor != 0:
13         wybor = int(input('Wybierz opcje (0-wyjscie, 1-mnozenie): '))
14
15         if wybor == 1:
16             obsluz_mnozenie()
```

Rys. 7.2. Program wykorzystujący dwie funkcje (lub w zależności od użytego kontekstu: funkcję i procedurę)

Przy obecnej strukturze kodu, program rozpoczyna swoją pracę dopiero od linii 11 – kod funkcji jest pomijany do momentu ich bezpośredniego wywołania z programu (linia 16). Po



Fundusze Europejskie  
Wiedza Edukacja Rozwój



Rzeczypospolita  
Polska



wejściu do pętli użytkownik proszony jest o podanie jednej z dwóch możliwych opcji. W przypadku wybrania operacji mnożenia, program wykonuje skok do funkcji (procedury) `obsluz_mnozenie`, gdzie pobiera od użytkownika dwie kolejne wartości i zamienia je na postać liczbową. Po zamianie wartości wywoływana jest dwulinijkowa funkcja `pomnoz`, która zwraca wynik mnożenia przekazanych wartości bezpośrednio do funkcji `print`. Po wyjściu z procedury program ponownie zwraca się z prośbą o wybranie jednej z dostępnych opcji.

#### Kącik autora

Warto przetestować działanie trybu pracy krokowej przy pracy z funkcjami.

Dodatkowym aspektem wartym omówienia jest przypadek, w którym programista w wyniku działania funkcji (lub procedury) chciałby otrzymać więcej niż jeden wynik. W języku Python istnieje możliwość zwracania z poziomu funkcji kilku wartości jednocześnie (zostaną wtedy zamienione automatycznie w listę) – wystarczy oddzielić wartości przecinkiem podczas zwracania po słowie `return`. Również dekompozycja wyniku jest niezwykle intuicyjna.

Powyższą sytuację zilustruje przykład funkcji, która będzie miała za zadanie jednocześnie zwrócić część całkowitą oraz resztę z dzielenia jednej liczby przez drugą (rys. 7.3)

```
1  def podziel_z_reszta(dzielna, dzielnik):
2      wynik_calkowity = int(dzielna / dzielnik)
3      reszta = dzielna % dzielnik
4
5      return wynik_calkowity, reszta
6
7  def obsluz_dzielenie():
8      x = int(input('Podaj dzielna: '))
9      y = int(input('Podaj dzielnik: '))
10
11     if y != 0:
12         wynik, reszta = podziel_z_reszta(x, y)
13         print('Wynik dzielenia: ' + str(wynik) + ' reszty ' + str(reszta))
14     else:
15         print('Dzielenie przez zero!')
16
17     wybor = 1
18     while wybor != 0:
19         wybor = int(input('Wybierz opcje (0-wyjście, 1-dzielenie): '))
20
21     if wybor == 1:
22         obsluz_dzielenie()
```

Rys. 7.3. Użycie funkcji zwracającej więcej niż jedną wartość

Program prezentuje się analogicznie do tego z rys. 7.2, z tą różnicą, że mnożenie zostało zastąpione dzieleniem z resztą. Po podaniu dzielnej i dzielnik wywoływana jest funkcja `podziel_z_reszta`, która oblicza wynik całkowity oraz resztę z dzielenia. Obie wartości zwracane są instrukcją `return` wykorzystując mechanizm kompozycji zmiennych. Mechanizm dekompozycji widoczny jest w linii nr 12, gdzie dwie zwrócone wartości są



Fundusze Europejskie  
Wiedza Edukacja Rozwój



Rzeczpospolita  
Polska



Unia Europejska  
Europejski Fundusz Społeczny

ponownie przypisywane pod zmienne: wynik oraz reszta. Metodę tą można rozszerzyć o dowolną liczbę parametrów do zwrócenia.

#### 4. Przykład z użyciem procedur i funkcji

W celu poglądowego zobrazowania użycia procedur i funkcji użyty zostanie stosunkowo niewielki przykład w postaci kalkulatora o ograniczonej funkcjonalności.

#### Treść zadania

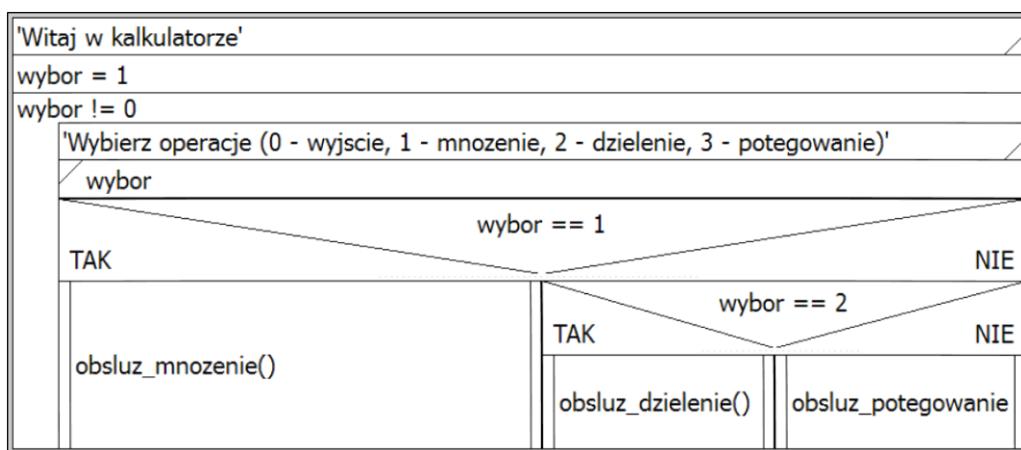
Zaprojektować kalkulator posiadający następującą funkcjonalność:

- mnożenie,
- dzielenie,
- potęgowanie

Obsługę każdej z wymienionych funkcjonalności umieścić w odrębnej procedurze. Wybór operacji powinien być umieszczony w pętli z możliwością jej opuszczenia.

#### Rozwiązania zadania

Treść zadania pozostawia sporo swobody w projektowaniu algorytmu aplikacji, jednak wymusza zastosowanie odrębnych procedur do obsługi poszczególnych działań. Mając na uwadze powyższe uwarunkowania, dość szybko można przejść do projektu głównej części diagramu (rys. 7.4).



Rys. 7.4. Główna część diagramu umożliwiająca przejście do kolejnych miejsc

Jak można zauważyć, zastosowanie procedur znaczaco uprościło organizację kodu na diagramie – przy każdym wyborze znajduje się wywołanie dokładnie jednej procedury. Wyjście z kalkulatora możliwe jest poprzez wybranie zera.

W tym momencie programista może przejść do opracowania metody obsługi poszczególnych działań. Jako pierwsze działanie wybrano mnożenie (procedura `obsluz_mnozenie`) – przykładową obsługę zaprezentowano na rys. 7.5.

```
procedure obsluz_mnozenie()
'Podaj pierwsza liczbe'
    liczba1
'Podaj druga liczbe'
    liczba2
wynik = liczba1 * liczba2
'Wynik operacji wynosi' + str(wynik)
```

Rys. 7.5. Podstawowa obsługa mnożenia umieszczona w odrębnej procedurze

Kod zawarty w obsłudze mnożenia nie powinien budzić większych wątpliwości, podobnie jak zawartość procedury `obsluz_dzielenie` przedstawionej na rys. 7.6.

```
procedure obsluz_dzielenie()
'Podaj pierwsza liczbe'
    liczba1
'Podaj druga liczbe'
    liczba2
liczba2 != 0
    TAK
        wynik = liczba1 / liczba2
        'Wynik operacji wynosi:' + str(wynik)
    NIE
        'Dzielenie przez zero!'
```

Rys. 7.6. Podstawowa obsługa dzielenia umieszczona w odrębnej procedurze

Ostatnią część programu stanowi obsługa potęgowania umieszczona na rys. 7.7.



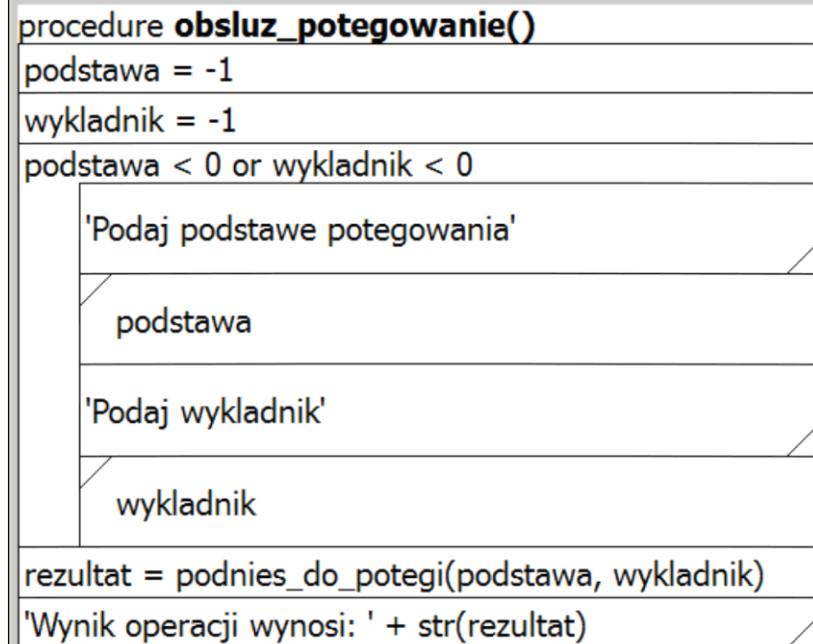
Fundusze  
Europejskie  
Wiedza Edukacja Rozwój



Rzeczpospolita  
Polska

Unia Europejska  
Europejski Fundusz Społeczny

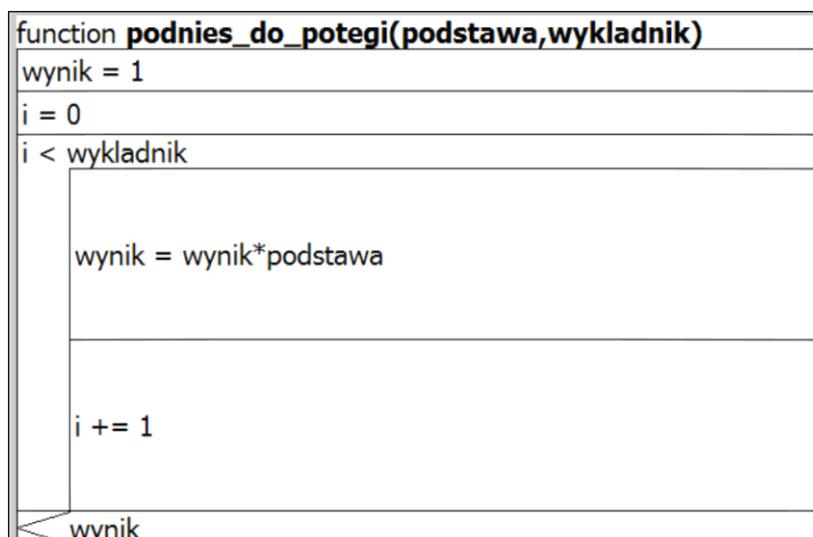




Rys. 7.7. Obsługa potęgowania umieszczona w odrębnej procedurze

Tym razem warto przyjrzeć się procedurze nieco dłużej. Mimo możliwości umieszczenia kompletnego kodu podnoszącego liczbę do potęgi, zdecydowano na rozbiście tego zadania na dwie części. Pierwszą z nich jest pobranie od użytkownika dwóch dodatnich wartości (kalkulator nie wspiera podnoszenia do potęgi ujemnej), by następnie wykorzystać pobraną podstawę i wykładnik do wywołania funkcji `podnies_do_potegi` w przedostatniej linii. Wywołanie funkcji i przypisanie zwracanej wartości realizowane jest w bloczku prostym. Otrzymany wynik wyświetlany jest następnie na ekranie.

Ostatnią część diagramu stanowi funkcja `podnies_do_potegi` – jej przykładową obsługę zawarto na rys. 7.8.



Rys. 7.8. Schemat funkcji `podnies_do_potegi`



Fundusze  
Europejskie  
Wiedza Edukacja Rozwój



Rzeczpospolita  
Polska

Unia Europejska  
Europejski Fundusz Społeczny



Funkcja `podnies_do_potegi` przyjmuje dwa parametry: podstawę oraz wykładnik – na ich podstawie wyliczany jest rezultat potęgowania, który jest ostatecznie zwracany przez funkcję (warto zwrócić uwagę na wcięcie w ostatniej linii symbolizujące zwrócenie wartości).

Schemat ten kończy omówienie reprezentacji procedur i funkcji z poziomu diagramów NS.

### Zadanie do realizacji

Zaimplementuj powyższe rozwiązanie w języku Python.

### Zadania do samodzielnej realizacji

1. Napisać program, który pozwoli użytkownikowi na wykonanie następujących opcji:
  - 1) dodanie nowej oceny z zakresu od 2 do 5 (wprowadzane wartości powinny być zapamiętane w tablicy)
  - 2) sprawdzenie, czy student ma zaliczony semestr (wszystkie oceny muszą być większe od 2)
  - 3) obliczenie średniej z podanych ocen (utworzyć funkcję `oblicz_srednia` i wywołać w programie głównym – funkcji należy przekazać tablicę ocen oraz jej aktualną długość)
  - 4) ustawienie wszystkich ocen większych od średniej po prawej stronie tablicy, mniejszych po lewej
  - 5) wyświetlenie wszystkich ocen
2. W oparciu o tablicę, utworzyć program pozwalający na następujące operacje: dodanie elementu na koniec listy, usunięcie elementu z końca listy, wyświetlenie wyniku, wyjście z programu.

Lista początkowo jest pusta. Wybór dokonywany jest przez użytkownika. W przypadku pierwszej operacji pobierana jest liczba. Wyświetlenie wyniku polega na wyświetleniu sumy wszystkich wartości na liście.

Ograniczenia: w jednej funkcji (również w programie głównym) można użyć tylko jednej instrukcji warunkowej (lub pętli).

3. Pobrać od użytkownika n wartości (wartości w przedziale od 1 do 100), znaleźć i wypisać dwie najmniejsze wartości, a następnie usunąć je z tablicy.

W programie utworzyć oraz wykorzystać:



**Fundusze Europejskie**  
Wiedza Edukacja Rozwój



**Rzeczpospolita Polska**



**Unia Europejska**  
Europejski Fundusz Społeczny

- funkcję `znajdzmin`, która jako parametr przyjmować będzie: tablicę, jej długość oraz wartość, od której będzie poszukiwać najmniejszej wartości (wszystkie mniejsze wartości będą ignorowane). Funkcja zwraca w wyniku znalezione minimum.
- procedurę `usun_elementy`, która jako parametr pobierze: tablicę, jej długość oraz wartość elementów do usunięcia (usunięcie elementu oznacza przesunięcie wszystkich elementów tablicy znajdujących się po prawej stronie usuwanego elementu do lewej strony).



**Fundusze  
Europejskie**  
Wiedza Edukacja Rozwój



**Rzeczpospolita  
Polska**

**Unia Europejska**  
Europejski Fundusz Społeczny



## ROZDZIAŁ 8. REKURENCJA

### Cel rozdziału:

Przedstawienie sposobu działania rekurencji.

### Zakres tematyczny:

1. Pojęcie rekurencji
2. Omówienie przykładu wykorzystujący rekurencję.

### Pytania kontrolne:

1. Czym jest funkcja?
2. Jaka jest różnica pomiędzy procedurą, a funkcją?
3. Czy jest parametr funkcji?

### 1. Pojęcie rekurencji

W rozdziale siódmym po raz pierwszy wprowadzono pojęcie funkcji – sekwencji instrukcji opatrzonej nadaną przez programistę nazwą. Funkcja w trakcie swojej pracy wykonuje kolejne instrukcje, by ostatecznie powrócić do miejsca, z którego została wywołana. Instrukcją wewnętrz pętli może być dowolna instrukcja – począwszy od prostego przypisania wartości, a skończywszy na wywołaniu innej procedury lub funkcji (jako przykład może posłużyć wywołanie funkcji `podnies_do_potegi` w procedurze `obsluz_potegowanie`, którą użyto na potrzeby budowy prostego kalkulatora).

Szczególnym przypadkiem wywołania jednej funkcji przez drugą jest sytuacja, w której funkcja wywołuje samą siebie. Proces ten nosi nazwę rekurencji – przykład zastosowania rekurencji przedstawiono na rys. 8.1 (diagram NS) oraz rys. 8.2 (język Python).

#### Kącik autora

Na początku może się wydawać się, że funkcja wywołująca samą siebie doprowadzi do zapętlenia się programu, bez możliwości jego zakończenia. Rzeczywiście, istnieje takie ryzyko – aby zapobiec temu zjawisku wykorzystuje się najczęściej instrukcję warunkową na początek funkcji, która sprawdza warunki ponownego wywołania samej siebie. Poprawne napisanie warunku w oparciu o dostarczony parametr jest kluczowe przy korzystaniu z rekurencji.

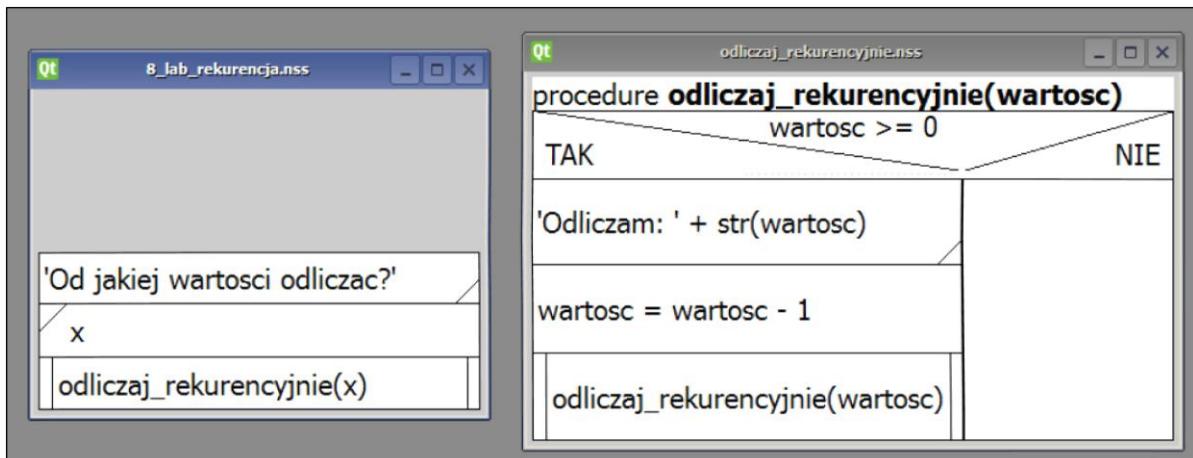


Fundusze Europejskie  
Wiedza Edukacja Rozwój



Rzeczpospolita  
Polska





Rys. 8.1. Prosta funkcja wykorzystująca rekurencję do odliczania kolejnych wartości

Warto zwrócić uwagę na fakt, iż w przedstawionym przekładzie nie wykorzystano konstrukcji pętli – odliczanie od podanej przez użytkownika wartości do zera wykonywane jest przy użyciu rekurencji.

```
1 def odliczaj_rekurencja(wartosc):
2     if wartosc >= 0:
3         print('Odliczam: ' + str(wartosc))
4         wartosc = wartosc-1
5         odliczaj_rekurencja(wartosc)
6
7 x = int(input('Od jakiej wartości odliczac? '))
8 odliczaj_rekurencja(x)
```

Rys. 8.2. Prosta funkcja wykorzystująca rekurencję do odliczania kolejnych wartości w języku Python

Na początku programu pobierana jest od użytkownika wartość, która jest następnie przekazywana do funkcji `odliczaj_rekurencja`. Kluczowym elementem funkcji rekurencyjnej jest definicja warunku kontynuacji (lub przerwania) – w przypadku nieprawidłowego doboru warunku, funkcja wpadnie w pętlę nieskończonych wywołań samej siebie. Sytuację tą można bardzo łatwo przetestować usuwając całkowicie instrukcję warunkową z ciała funkcji.

#### Kącik autora

Niechlujnie zaprojektowana rekurencja potrafi przysporzyć wielu problemów, głównie ze względu na znacznie utrudniony proces analizy i testowania kodu.



Fundusze  
Europejskie  
Wiedza Edukacja Rozwój



Rzeczpospolita  
Polska



Unia Europejska  
Europejski Fundusz Społeczny

## 2. Praktyczny przykład z wykorzystaniem rekurencji

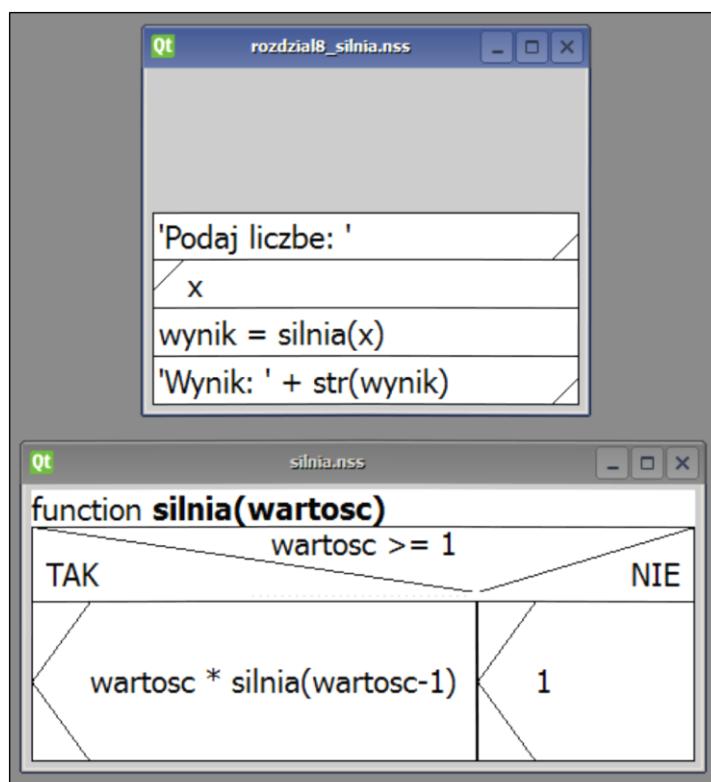
Do jednych z popularniejszych przykładów korzystających z własności rekurencji można zaliczyć takie przykłady jak:

- wyznaczanie ciągu Fibonacciego,
- sortowanie danych z użyciem algorytmu Quicksort,
- obliczanie wartości silni.

W rozdziale tym zaprezentowany zostanie ostatni przykład znajdujący się na liście, który stanowi poniekąd rozwinięcie przykładu z poprzedniego rozdziału. Dla przypomnienia, wartość silni z liczby stanowi iloczyn wszystkich kolejnych liczb całkowitych niewiększych niż wartość silni (wyjątkiem jest zero, dla którego wartość silni wynosi 1), na przykład:

$$5! = 5 * 4 * 3 * 2 * 1 = 120$$

Znając definicję silni można przystąpić do projektu funkcji (rys. 8.3)



Rys. 8.3. Schemat funkcji rekurencyjnej implementacji funkcji silnia

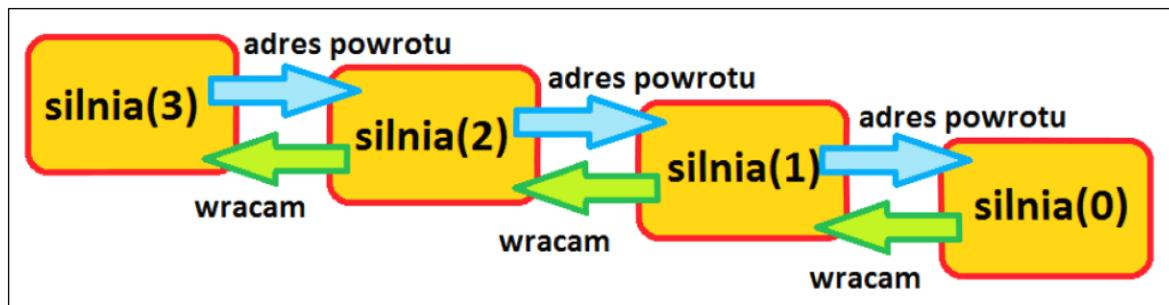
Tym razem do rekurencji wykorzystano funkcję ze względu na fakt użycia zwracanej wartości – funkcja `silnia` wywoływana jest do momentu osiągnięcia wartości równej zero, co kończy ciąg wywołań funkcji (jedynie dla wartości większej bądź równej jeden wywoływana jest ponownie funkcja `silnia`).

Analogiczną implementację funkcji silnia w języku Python przedstawiono na rys. 8.4.

```
1 def silnia(wartosc):  
2     if wartosc >= 1:  
3         return wartosc * silnia(wartosc - 1)  
4     else:  
5         return 1  
6  
7  
8     x = int(input('Podaj liczbe: '))  
9     wynik = silnia(x)  
10    print(str(x) + '! = ' + str(wynik))
```

Rys. 8.4. Implementacja funkcji silnia w języku Python

W tym momencie warto zwrócić uwagę na jeszcze jeden aspekt rekurencji – za każdym razem, gdy wywoływana jest kolejna funkcja lub procedura, program musi każdorazowo zapamiętać adres powrotu. Im głębsza rekurencja, tym więcej adresów musi zostać zapamiętanych, aby aplikacja mogła powrócić do pierwszego miejsca wywołania – sytuacja ta została zaprezentowana na rys. 8.5.



Rys. 8.5. Mechanizm odkładania adresu powrotu podczas funkcji rekurencyjnych

Jako, że środowisko uruchomieniowe dysponuje ograniczonym miejscem do przechowywania adresów powrotnych, wykonanie głębokiej rekurencji może zakończyć się błędem przekroczenia tej wartości. Przedstawioną sytuację można sprawdzić wywołując funkcję silnia dla nieco większej wartości. Fakt ten sprawia, iż zwykle w przypadku wyboru pomiędzy algorytmem rekurencyjnym, a iteracyjnym (wykorzystującym pętle) wykonującym to samo zadanie, wykorzystywany jest algorytm iteracyjny.

Kącik autora

Spróbuj sprawdzić, od jakiej wartości program przestanie działać!



Fundusze Europejskie  
Wiedza Edukacja Rozwój



Rzeczpospolita  
Polska



Unia Europejska  
Europejski Fundusz Społeczny

## **Zadanie**

Prześledzić działanie funkcji silnia korzystając z trybu pracy krokowej. Sprawdzić dla jakiej wartości przekazanej przez użytkownika aplikacja przestanie działać prawidłowo.



**Fundusze  
Europejskie**  
Wiedza Edukacja Rozwój



**Rzeczpospolita  
Polska**

**Unia Europejska**  
Europejski Fundusz Społeczny





Zintegrowany  
Program  
Rozwoju  
Politechniki  
Lubelskiej -  
część druga

Materiały zostały opracowane w ramach projektu  
„Zintegrowany Program Rozwoju Politechniki Lubelskiej – część druga”,  
umowa nr **POWR.03.05.00-00-Z060/18-00**  
w ramach Programu Operacyjnego Wiedza Edukacja Rozwój 2014-  
2020  
współfinansowanego ze środków Europejskiego Funduszu  
Społecznego



Fundusze  
Europejskie  
Wiedza Edukacja Rozwój



Rzeczpospolita  
Polska



Unia Europejska  
Europejski Fundusz Społeczny