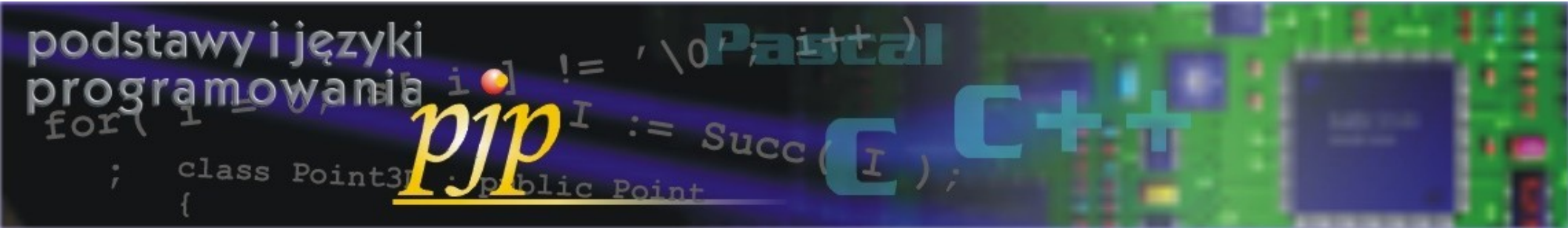
Programowanie I

Podstawy programowania

Podejście strukturalne

Roman Simiński
roman.siminski@us.edu.pl
roman@siminskionline.pl

Pierwszy program



Przelicznik jednostek miar

Problem

Proszę napisać program, pozwalający na przeliczenie odległości wyrażonej w milach amerykańskich na kilometry. Odległość powinna być nieujemną liczbą rzeczywistą.

Wiedza dziedzinowa, czyli jak to przeliczyć?

Przeliczniki, na wszelki wypadek w obie strony:

$$\text{Odległość w milach} = 0.625 \cdot \text{odległość w kilometrach}$$

$$\text{Odległość w kilometrach} = 1.6 \cdot \text{odległość w milach}$$

Przelicznik jednostek miar – po co taki program...

W TAKICH samochodach nie ma komputerów pokładowych, i zwykle wszystko wyskalowane jest w milach, warto jednak wiedzieć ile to w kilometrach...

Obrazy pochodzą ze strony: https://www.topspeed.com/best-classic-american-muscle-cars/

Przelicznik jednostek miar - wymagania

Zadaniem programu jest zapytanie użytkownika o odległość wyrażoną w milach, wczytanie wartości podanej przez użytkownika oraz wyświetlenie wartości odległości wyrażonej w kilometrach.

```
Przeliczam odległość wyrażoną w milach na kilometry
Podaj odległość w milach: 55
W kilometrach to: 88.00
```

Przelicznik jednostek miar – wymagania, cd...

Warto pomyśleć o błędach w danych wejściowych:

```
Przeliczam odległość wyrażoną w milach na kilometry
Podaj odległość w milach: ala ma kota
Błędna liczba
```

```
Przeliczam odległość wyrażoną w milach na kilometry
Podaj odległość w milach: -100
Odległość powinna być nieujemna
```

Przelicznik jednostek miar – wymagania, cd...

Warto pomyśleć o błędach w danych wejściowych:

```
Przeliczam odległość wyrażoną w milach na kilometry
Podaj odległość w milach: ala ma kota
Błędna liczba
```

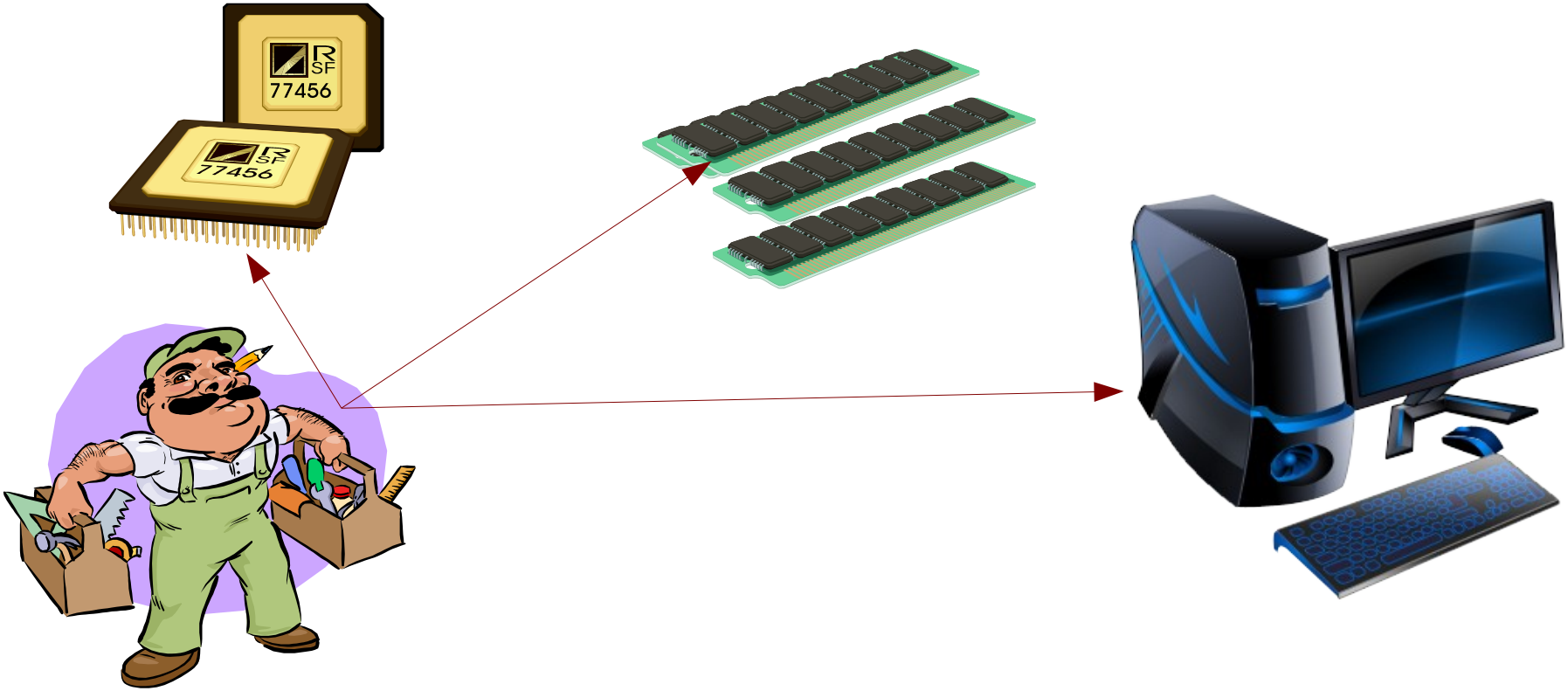
```
Przeliczam odległość wyrażoną w milach na kilometry
Podaj odległość w milach: -100
Odległość powinna być nieujemna
```

Jak się za to zabrać? Załóżmy, że nic nie wiemy o programowaniu...

Spojrzenie użytkownika — komputer to czarna skrzynka

Użytkownika interesuje CO potrafi zrobić komputer a nie JAK to robi... .

Spojrzenie programisty — komputer od środka



Programista musi wiedzieć jak zmusić komputer, żeby zrobił to co trzeba

On musi wiedzieć JAK to się robi

Algorytm, czyli JAK TO SIĘ ROBI!

Programista musi znać sposób rozwiązania problemu, który stanowi przedmiot programu. Ten sposób określa algorytm.

Algorytm określa *dane* oraz skończony ciąg *operacji*, jakie należy na tych danych wykonać, aby rozwiązać dowolny problem z określonej klasy problemów

Inaczej mówiąc, algorytm określa *dane* oraz *plan akcji*, jakie należy wykonać, aby osiągnąć założony cel

Metody zapisu algorytmów

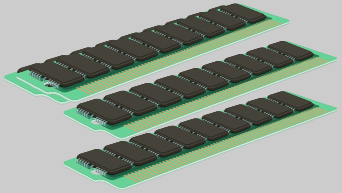
- Język naturalny (potoczny) określający ciąg kroków algorytmu.
- Notacje graficzne, najpopularniejsze są schematy blokowe.
- Pseudokod — połączenie języka naturalnego i notacji matematycznej z elementami języków programowania.
- Zapis w postaci kodu programu w wybranym języku programowania.

Co to za „dane”, co to za „operacje”?

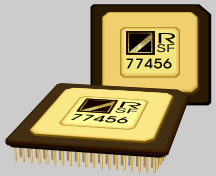
Dane, które program pamięta w czasie swojego działania

Wykonanie programu, obliczenia, sterowanie, powtórzenia, decyzje itp., itd. ...

Pamięć operacyjna



Procesor i spółka



Pamięć zewnętrzna

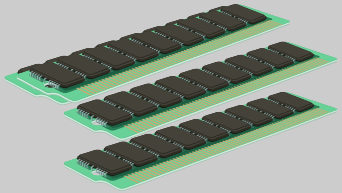
Wejście i wyjście

Dane, które mają być zapamiętane trwale

Operacje wejścia-wyjścia

A co z przeliczaniem z kilometrów na mile?

Pamięć operacyjna



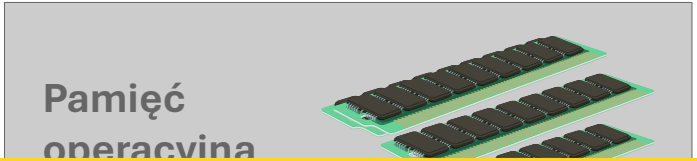
55

Odczyt ze strumienia wejściowego programu - klawiatura

55

55 mil,
ile to kilometrów?

A co z przeliczaniem z kilometrów na mile?



W jaki sposób można w pamięci operacyjnej przechować liczbę wprowadzoną z klawiatury?

Odczyt ze strumienia wejściowego programu - klawiatura

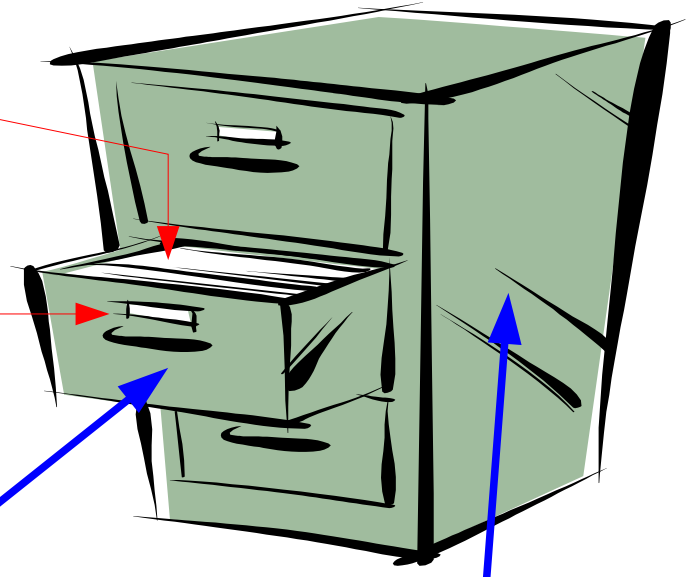
100

Jak dane są przechowywane w pamięci operacyjnej?

Zmienna jest to element programu, rezydujący w pamięci operacyjnej, przeznaczony do przechowywania wartości

Każda zmienna przechowuje wartość ustalonego typu

Każda zmienna ma swoją nazwę



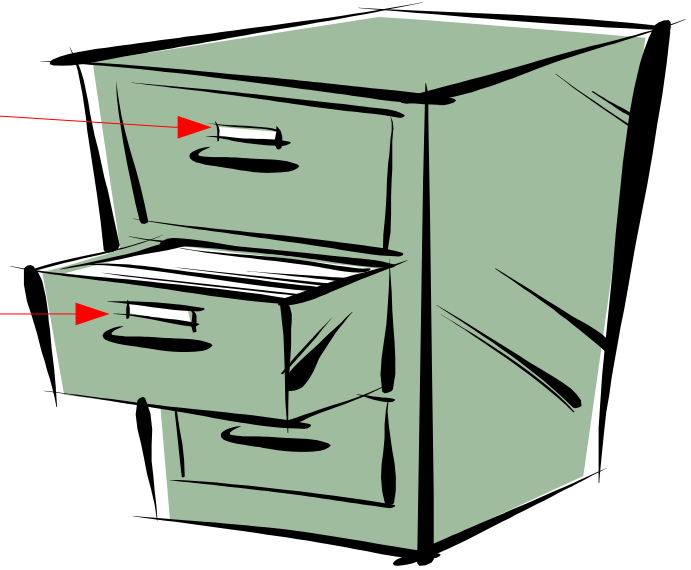
Zmienna

Pamięć operacyjna

Rola nazwy zmiennej

Nazwa zmiennej jednoznacznie identyfikuje ją w programie
Nazwa zmiennej jest unikatowa w zasięgu zmiennej
Posługując się nazwą zmiennej, posługujemy się zapisaną w niej wartością

Każda zmienna ma swoją nazwę



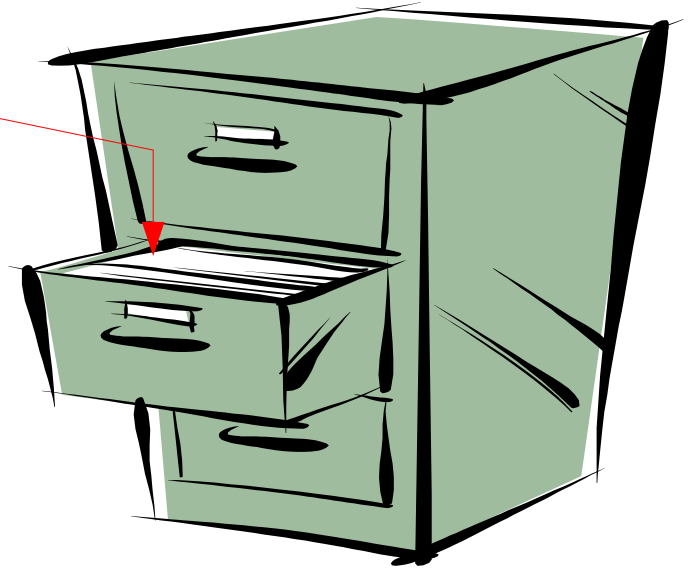
Nazwa zmiennej:

- To słowo, skrót, akronim lub ich zbitek, wymyślony przez programistę.
- Zwykle nazwa zmiennej może zawierać tylko litery, cyfry i znaki podkreślenia i musi rozpoczynać się od litery.

Rola typu zmiennej

Typ zmiennej określa jakie wartości można do niej wstawiać

Każda zmienna ma swój typ



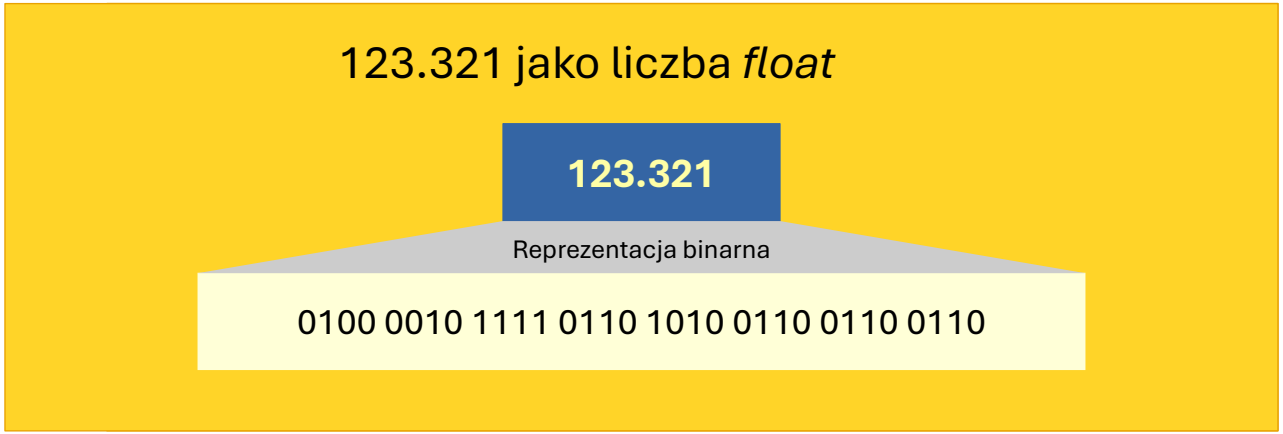
Dokładniej, typ zmiennej określa:

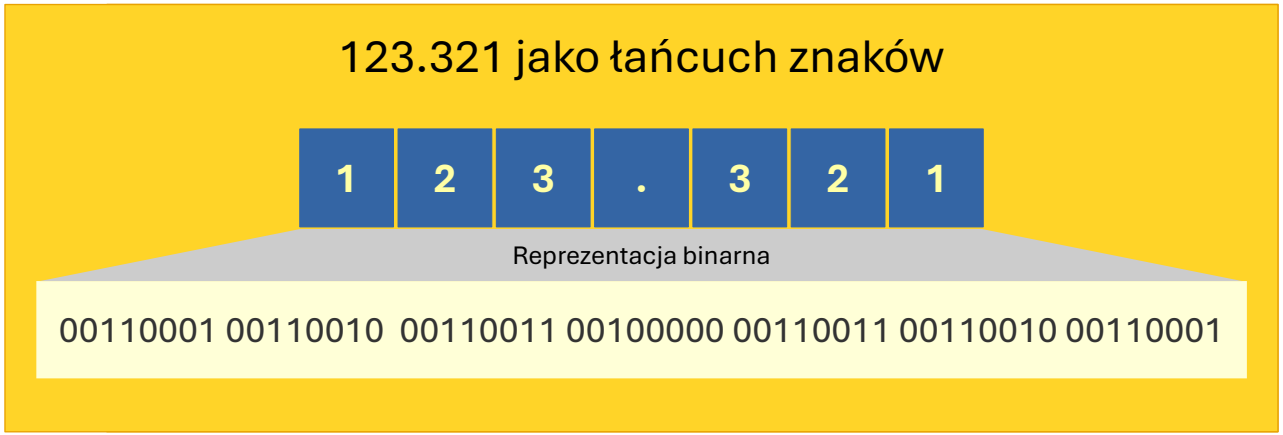
- Jakie wartości może przyjmować konkretna zmienna, i z jakiego zakresu mogą one pochodzić.
- Jakie operacje można wykonywać na zmiennej.
- Ile bajtów zmienna zajmuje w pamięci operacyjnej.

Podstawowe typy zmiennych

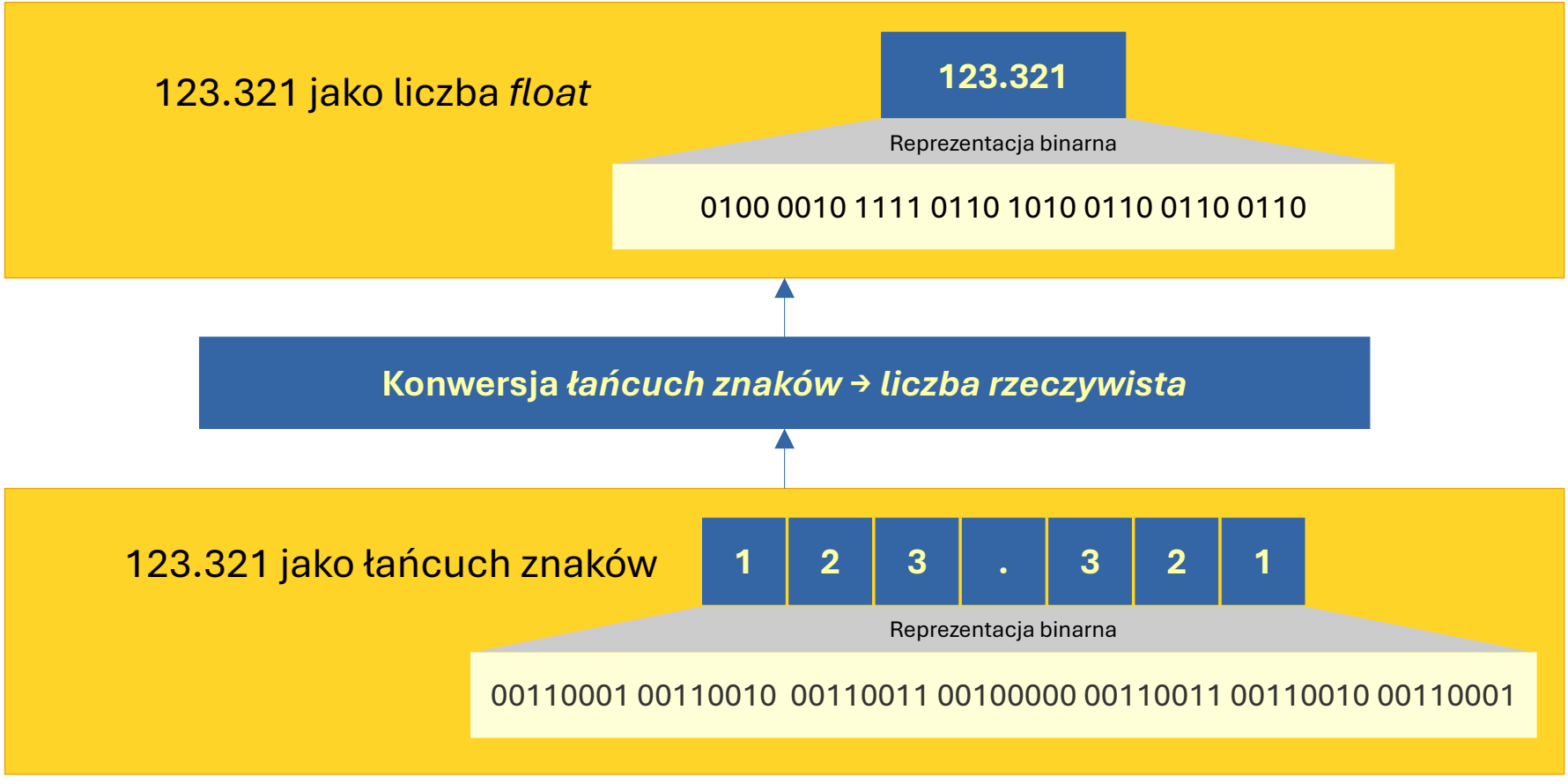
Typy występujące we większości języków:

- Liczba całkowita – *integer*, *int*.
- Liczba rzeczywista – *real*, *float*, *double*.
- Znak – *char*.
- Łańcuch znaków – *string*.
- Wartość logiczna – bool, boolean.

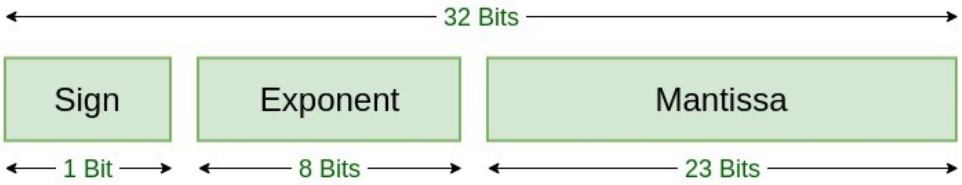




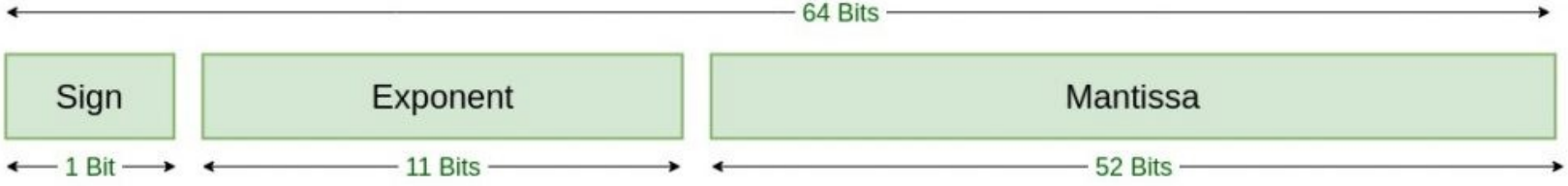
Konwersje typów zmiennych - przykład



Liczby rzeczywiste – float i double



Single Precision
IEEE 754 Floating-Point Standard



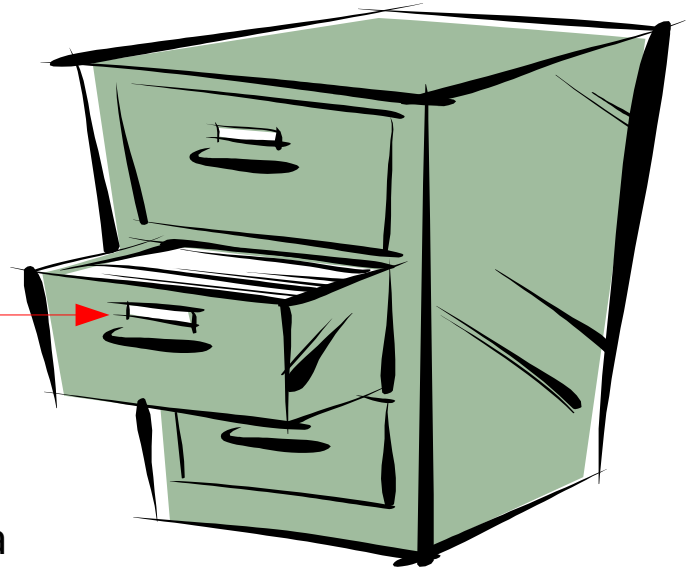
Double Precision
IEEE 754 Floating-Point Standard

https://www.geeksforgeeks.org/ieee-standard-754-floating-point-numbers/

Zmienne a przeliczenie odległości

Potrzebujemy miejsca w pamięci operacyjnej, w którym przechowamy na czas działania programu odległość w milach, podaną przez użytkownika

Potrzebujemy zatem zmiennej,
zdolnej do przechowania
liczby rzeczywistej,
niech ta zmienna
nazywa się **mile**

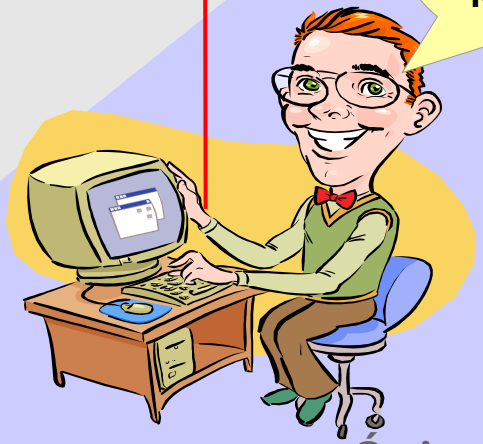


Po co? Bo chcemy zapamiętaną w zmiennej *mile* wartość użyć w wyrażeniu przeliczającym odległość na kilometry

Kolejne czynności programu

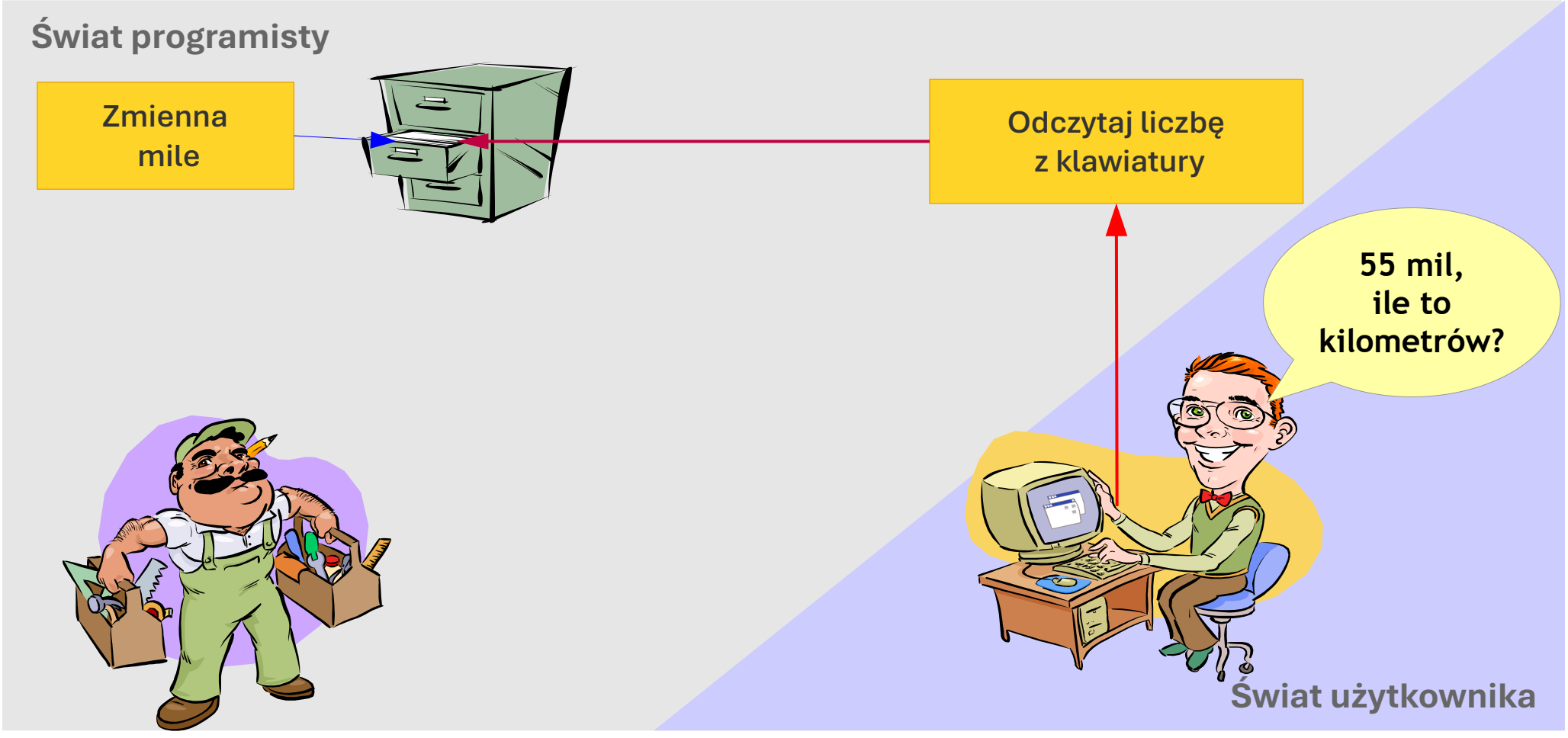
Świat programisty

Odczytaj liczbę
z klawiatury

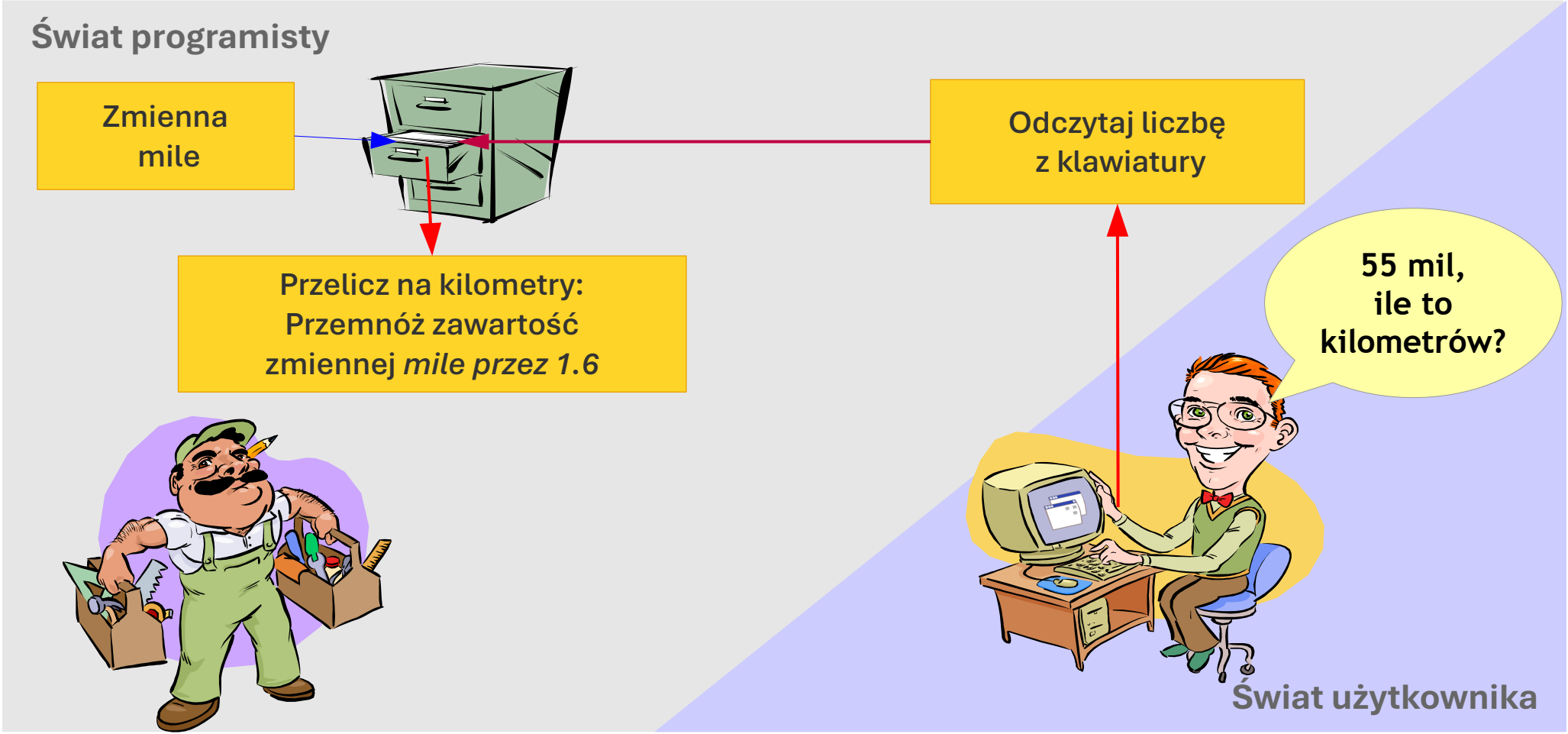


Świat użytkownika

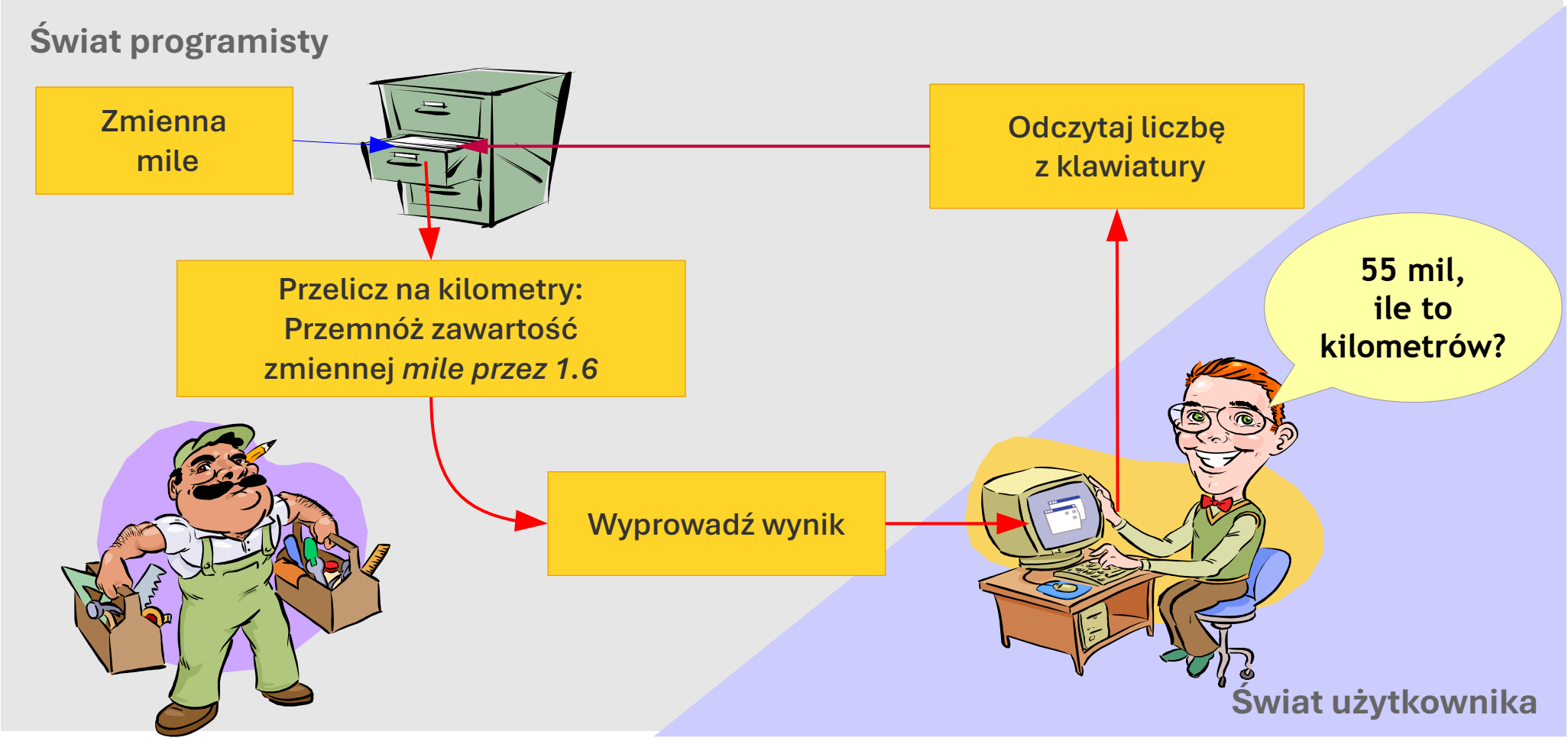
Kolejne czynności programu



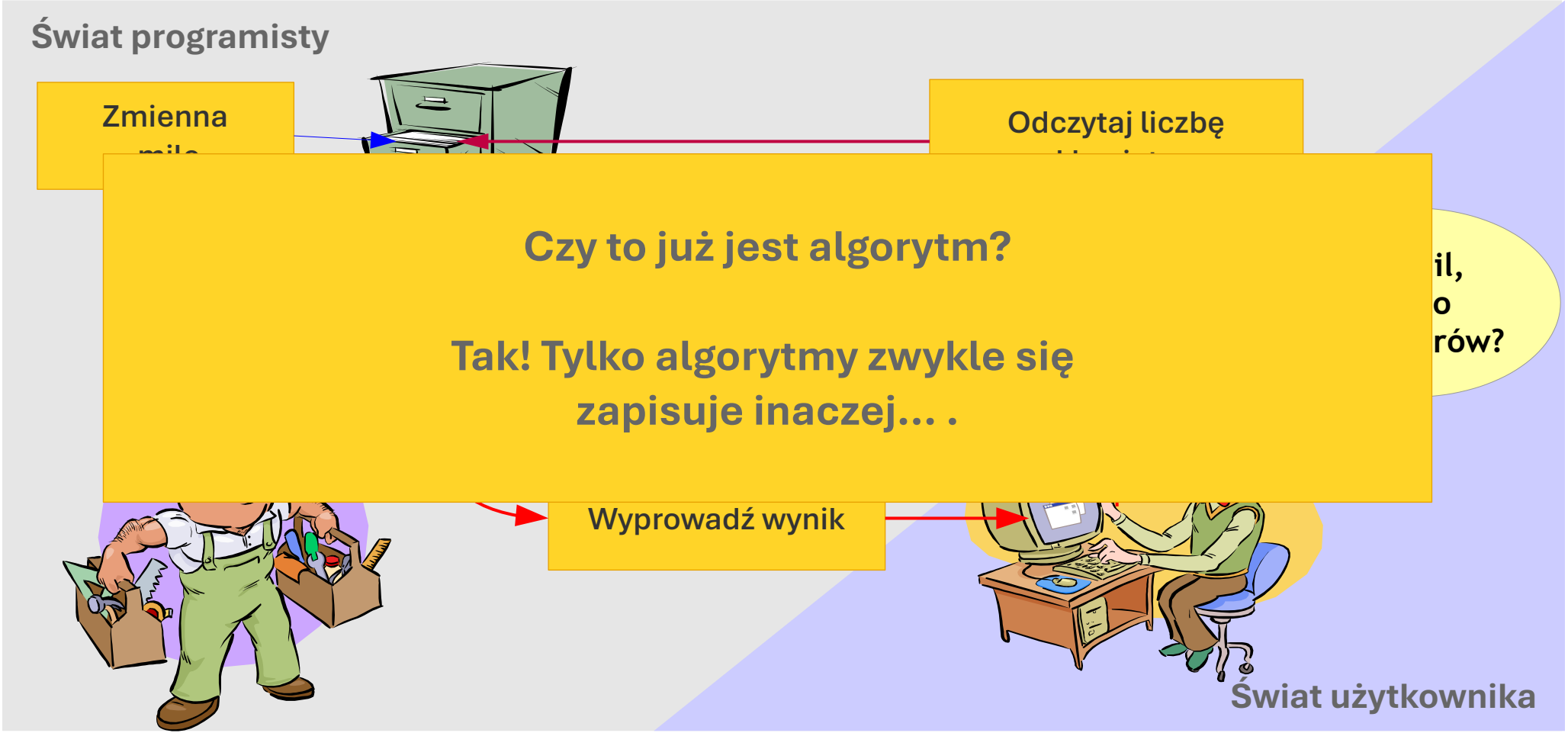
Kolejne czynności programu



Kolejne czynności programu



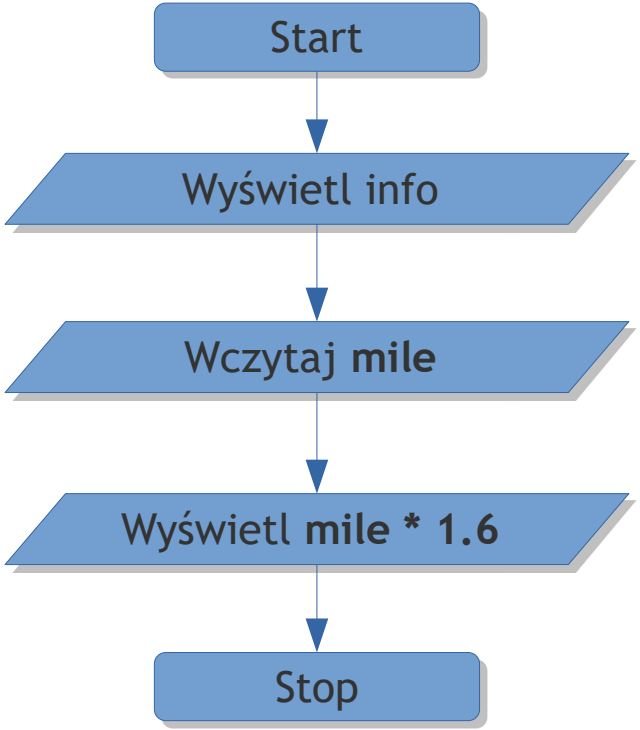
Kolejne czynności programu



Algorytm przeliczenia odległości — opis słowny

- Niech **mile** będzie zmienną typu liczba rzeczywista.
- Wyświetl informacje o programie.
- Wyświetl tekst zachęty do wprowadzenia odległości w milach.
- Odczytaj liczbę, zapisz w zmiennej **mile**.
- Przelicz na kilometry i wyprowadź: **mile** * 1.6.

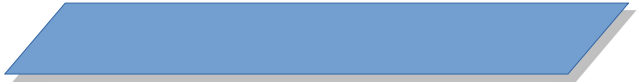
Algorytm przeliczenia odległości — schemat blokowy



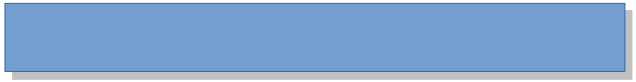
Podstawowe elementy schematów blokowych

Bloki początku i końca algorytmu:

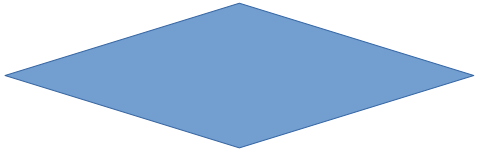
Blok operacji wejścia/wyjścia



Blok wykonawczy



Blok decyzyjny



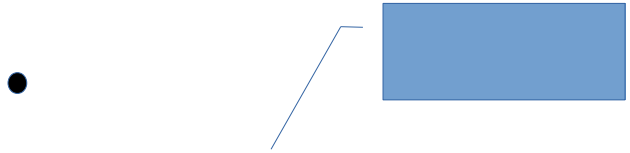
Inne elementy schematów blokowych

Blok operacji wykonywanej poza rozważanym algorytmem

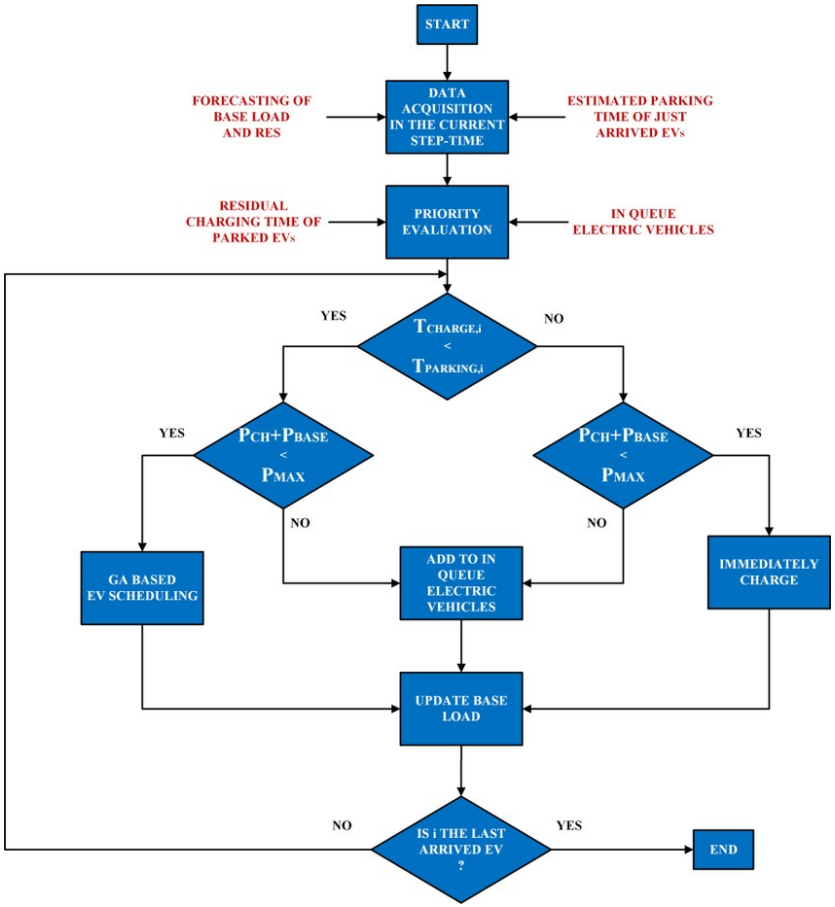
Łączniki stronnicowe

Łącznik międzystronnicowy

Punkt łączący i komentarz objaśniający



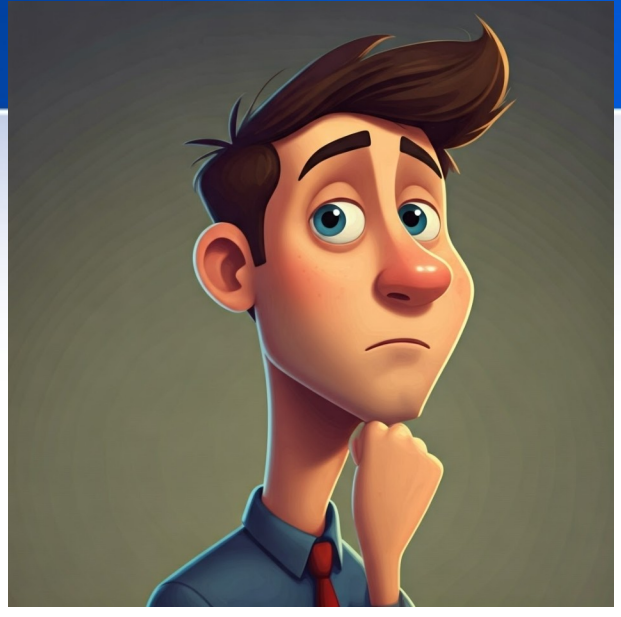
Przykładowy schemat blokowy



https://www.researchgate.net/figure/Scheduling-Algorithm-Block-Diagram_fig37_303629402

Znamy algorytm, zatem napiszmy program...

Ale jeszcze jedna sprawa...



Zanim napiszemy pierwszy program...

- Przypomnijmy sobie co to jest funkcja.
- Funkcje w programowaniu przypominają nieco funkcje matematyczne.
- Definicja funkcji określa jej *nazwę*, *parametry* (*argumenty*) oraz *ciało*.

$$g(x) = 2x + 5$$

Zanim napiszemy pierwszy program...

- Zdefiniowana funkcja, to coś, co potencjalnie może zadziałać.
- Aby funkcja zadziałała, należy jej użyć, inaczej mówiąc wywołać.

$$g(x) = 2x + 5$$

Nazwa funkcji

Parametr formalny

Ciało funkcji

Wywołanie funkcji

- Zdefiniowana funkcja, to coś, co potencjalnie może zadziałać.
- Aby funkcja zadziałała, należy jej użyć, inaczej mówiąc wywołać.

$$g(x) = 2x + 5$$

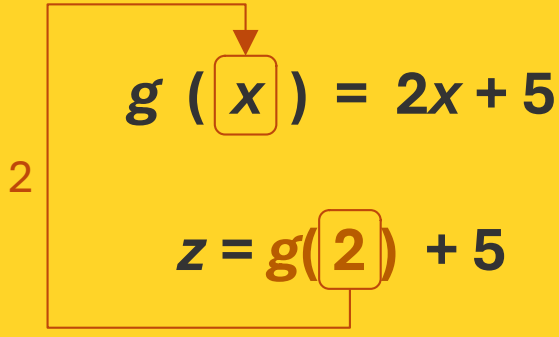
$$z = \boxed{g(2)} + 5$$

- Wywołanie funkcji powoduje wykonanie jej ciała oraz udostępnienie w miejscu wywołania rezultatu funkcji.
- Rezultat funkcji w matematyce ma zwykle wartość liczbową.

Wywołanie funkcji g(x)
Liczba 2 to
parametr aktualny
wywołania funkcji

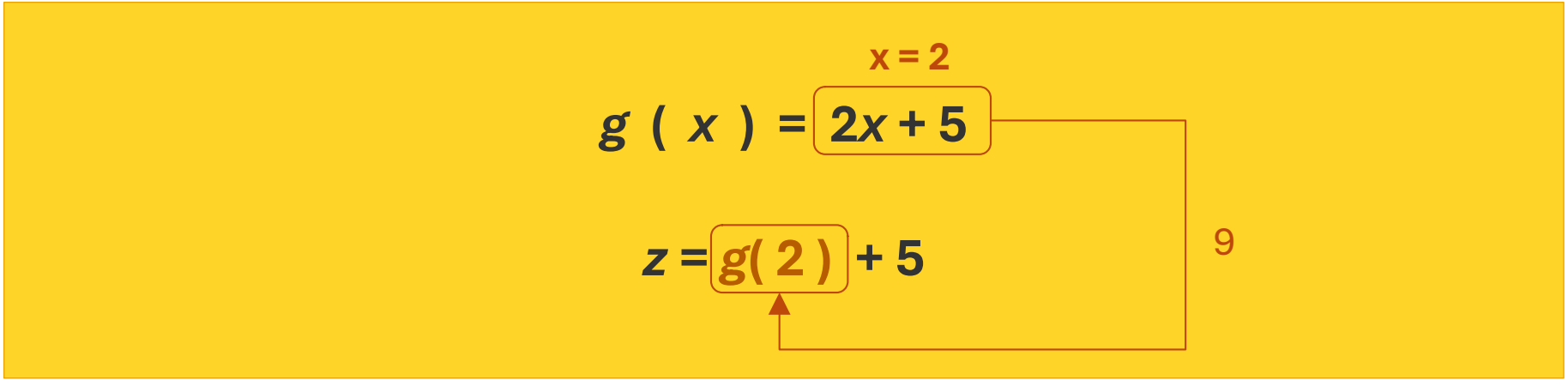
Przekazywanie parametrów

- W trakcie wywołania funkcji parametr aktualny wywołania, przekazywany jest do parametru formalnego funkcji.



Wykonanie ciała funkcji

- Uaktywnienie przepisu na wykonanie funkcji ze skonkretyzowaną wartością parametru formalnego.



Ostatecznie otrzymujemy rezultat wywołania

- W miejscu wywołania funkcji otrzymujemy jej rezultat – wartość wypracowaną przez ciało funkcji z wykorzystaniem parametru formalnego.

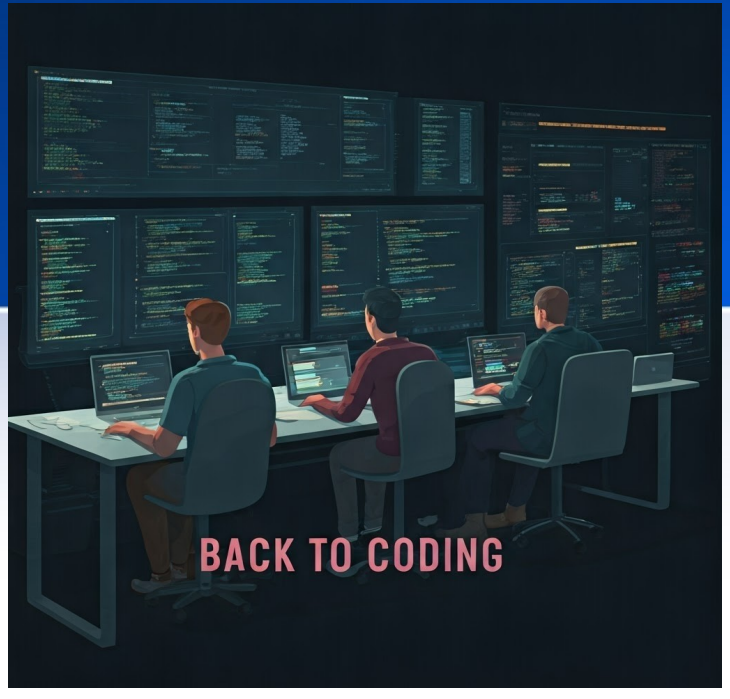
$$g(x) = 2x + 5$$

$$z = 9 + 5$$

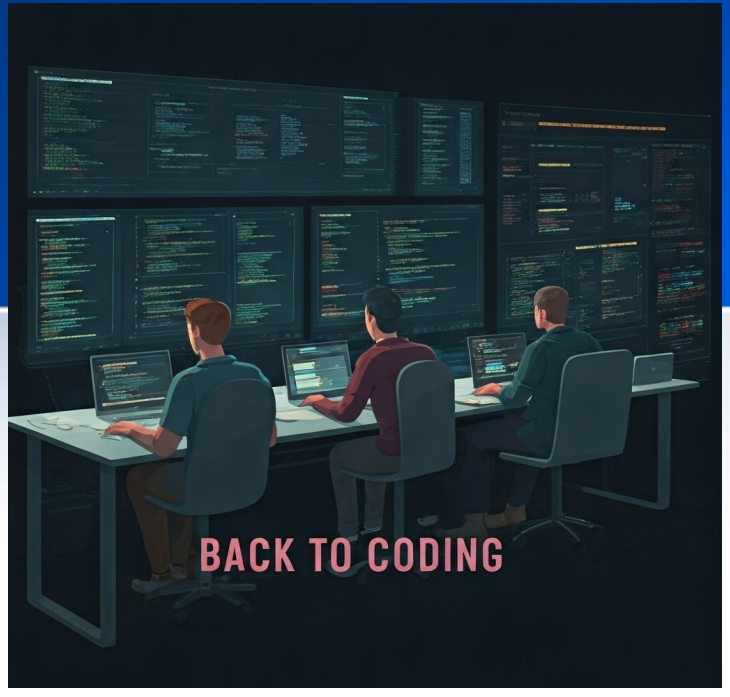
Zapamiętajmy...

- Funkcja to coś, co posiada swoją nazwę.
- Używając tej nazwy, można funkcję wywołać.
- Podczas wywołania funkcji następuje opracowanie parametrów.
- Opracowanie to polega na przekazaniu wartości parametrów aktualnych wywołania funkcji do odpowiednich parametrów formalnych.
- Wywołanie funkcji powoduje wykonanie jej ciała.
- W matematyce to zwykle wyznaczenie pewnej wartości liczbowej, która jest rezultatem funkcji.
- Ciało funkcji określa ciąg operacji, które należy wykonać aby wyznaczyć rezultat funkcji.
- Rezultat funkcji jest udostępniany w miejscu wywołania, „komuś”, kto tę funkcję wywołał.

W końcu...



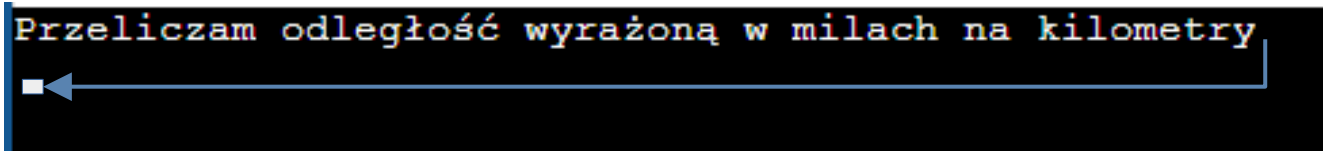
Python



Zacznijmy od języka Python – wyprowadzanie informacji

- Polecenie **print** wyprowadza dane do strumienia wyjściowego programu skojarzonego z konsolą. Kursor konsoli jest domyślnie przenoszony na początek następnej linii.

```
print("Przeliczam odległość wyrażoną w milach na kilometry")
```



- "Przeliczam odległość wyrażoną w milach na kilometry" – to literał łańcuchowy.
- Cudzysłowy to ograniczniki napisu, nie są jego częścią.

Literały łańcuchowe

- W języku Python można stosować zamiennie parami ograniczniki w postaci cudzysłowów lub apostrofów.
- Pozwala to na stosowaniu w łańcuchach znaków symboli cudzysłowów i apostrofów:

"Łańcuchy znaków żargonowo nazywa się ‘stringami’..."

‘Łańcuchy znaków żargonowo nazywa się ”stringami”..’

- Można również stosować w napisach znaki specjalne:

"Łańcuchy znaków żargonowo nazywa się \”stringami\”..."

‘Łańcuchy znaków żargonowo nazywa się \’stringami\’..’

Python – wprowadzanie informacji

- Funkcja **input** pozwala na wczytanie se strumienia wejściowego programu skojarzonego z klawiaturą dowolnego ciągi znaków, zatwierdzonego naciśnięciem klawisza Enter.

```
print("Przeliczam odległość wyrażoną w milach na kilometry")
mile_jako_tekst = input("Podaj odległość w milach: ")
```

- Rezultatem funkcji **input** jest łańcuch znaków wprowadzony przez użytkownika.
- Łańcuch znaków będący parametrem funkcji input wyświetlany jest jako tekst zachęty do wprowadzenia przez użytkownika danych.

```
Przeliczam odległość wyrażoną w milach na kilometry
Podaj odległość w milach:
```

```
Przeliczam odległość wyrażoną w milach na kilometry
Podaj odległość w milach: 55
```

Python – pierwsza zmienna

- Funkcja **input** pozwala na wczytanie se strumienia wejściowego programu skojarzonego z klawiaturą dowolnego ciągi znaków, zatwierdzonego naciśnięciem klawisza Enter.

```
print("Przeliczam odległość wyrażoną w milach na kilometry")
mile_jako_tekst = input("Podaj odległość w milach: ")
```

- Interpreter języka Python tworzy dynamicznie zmienną **mile_jako_tekst**, ustalając, że jest to zmienna przeznaczona do zapamiętywania łańcuchów znaków.
- Dokumentacja funkcji **input** mówi:

input(*prompt*)

If the *prompt* argument is present, it is written to standard output without a trailing newline. The function then reads a line from input, converts it to a string (stripping a trailing newline), and returns that. When EOF is read, `EOFError` is raised. Example:

https://docs.python.org/3/library/functions.html#input

Python – konwersja do danej wejściowej do postaci liczby

- Funkcja **float()** dokonuje konwersji łańcucha znaków przekazanego parametrem na odpowiadającą temu łańcuchowi wartość w postaci liczby rzeczywistej.
- W programowaniu liczby rzeczywiste reprezentowane są w postaci zmiennoprzecinkowej, ang. *floating point*, stąd nazwa *float*.

```
print("Przeliczam odległość wyrażoną w milach na kilometry")
mile_jako_tekst = input("Podaj odległość w milach: ")
mile = float(mile_jako_tekst)
```

Uwaga – jeżeli użytkownik nie wprowadzi prawidłowej wartości liczbowej, funkcja **float()** przerwie awaryjnie swoje działanie.

```
Podaj odległość w milach: coś tam
Traceback (most recent call last):
  File "/home/main.py", line 3, in <module>
    mile = float(mile_jako_tekst)
ValueError: could not convert string to float: 'coś tam'
```

Python – wyznaczenie i wyprowadzenie wyniku

- Można teraz użyć zmiennej **mile** do wyliczenia odległości w kilometrach.

```
print("Przeliczam odległość wyrażoną w milach na kilometry")
mile_jako_tekst = input("Podaj odległość w milach: ")
mile = float(mile_jako_tekst)
print(mile * 1.6)
```

```
Przeliczam odległość wyrażoną w milach na kilometry
Podaj odległość w milach: 55
88.0
```

- Tak prezentacja wyników nie jest jednak jasna i jednoznaczna.

Python – lepsze wyprowadzenie wyniku

- Funkcja **print()** może otrzymywać wiele parametrów oddzielanych przecinkami.

```
print("Przeliczam odległość wyrażoną w milach na kilometry")
mile_jako_tekst = input("Podaj odległość w milach: ")
mile = float(mile_jako_tekst)
print("W kilometrach to:", mile * 1.6)
```

```
Przeliczam odległość wyrażoną w milach na kilometry
Podaj odległość w milach: 55
W kilometrach to: 88.0
```

Python – uproszczenie

- Można zrezygnować ze zmiennej **mile_jako_tekst** używając kaskadowego wywołania funkcji **input i float.**

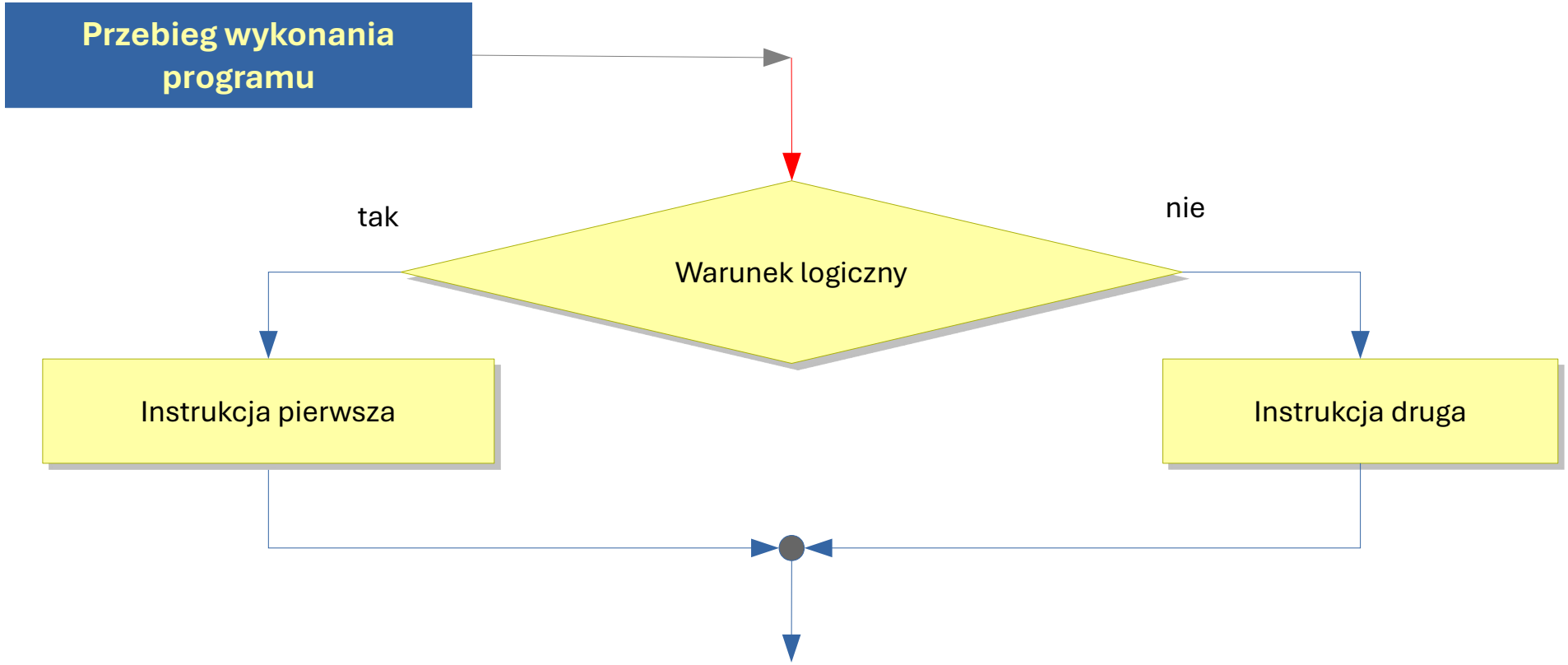
```
print("Przeliczam odległość wyrażoną w milach na kilometry")
mile = float(input("Podaj odległość w milach: "))
print("W kilometrach to:", mile * 1.6)
```

Python – zabezpieczenie przed ujemną wartością odległości

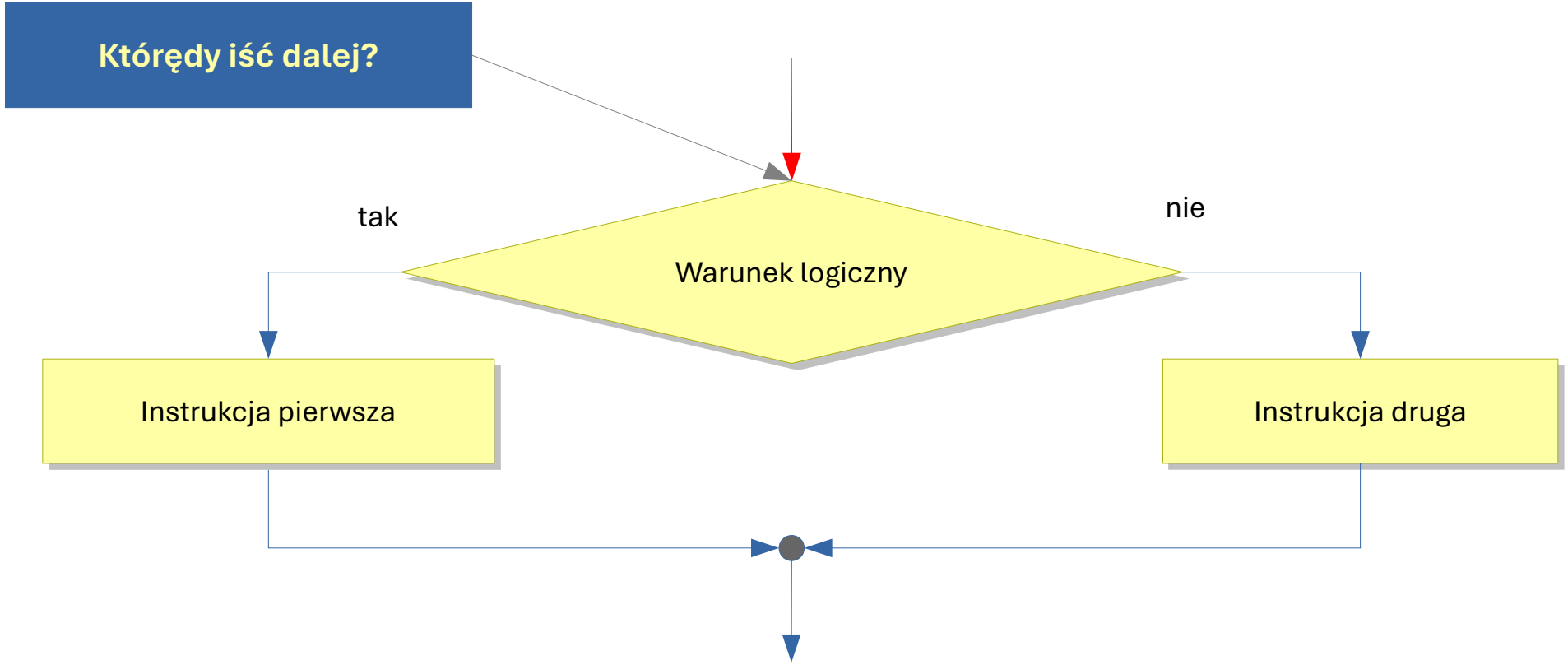
- Musimy oszacować wartość zmiennej **mile** i podjąć decyzję który komunikat wyprowadzimy.

```
print("Przeliczam odległość wyrażoną w milach na kilometry")
mile = float(input("Podaj odległość w milach: "))
???
    print("W kilometrach to:", mile * 1.6)
???
    print("Odległość powinna być nieujemna")
```

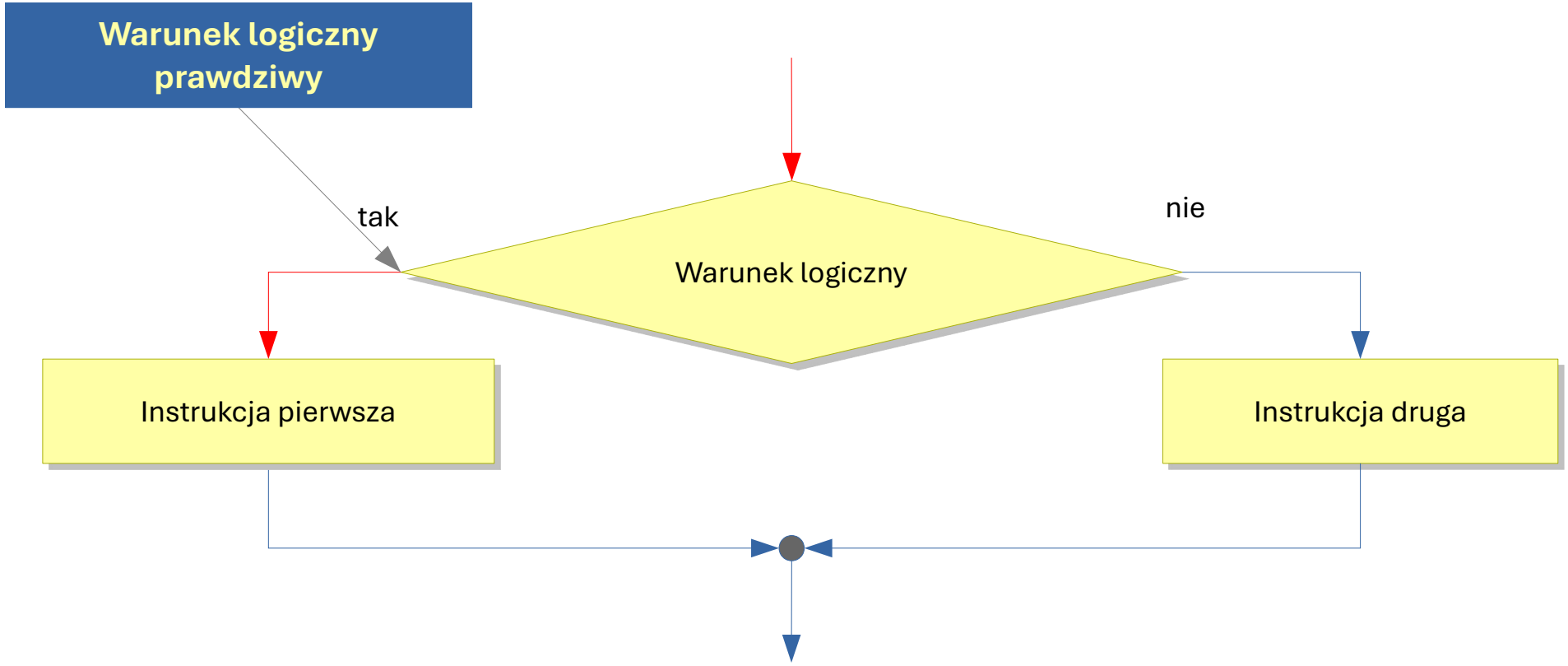
Instrukcja alternatywy – schemat blokowy



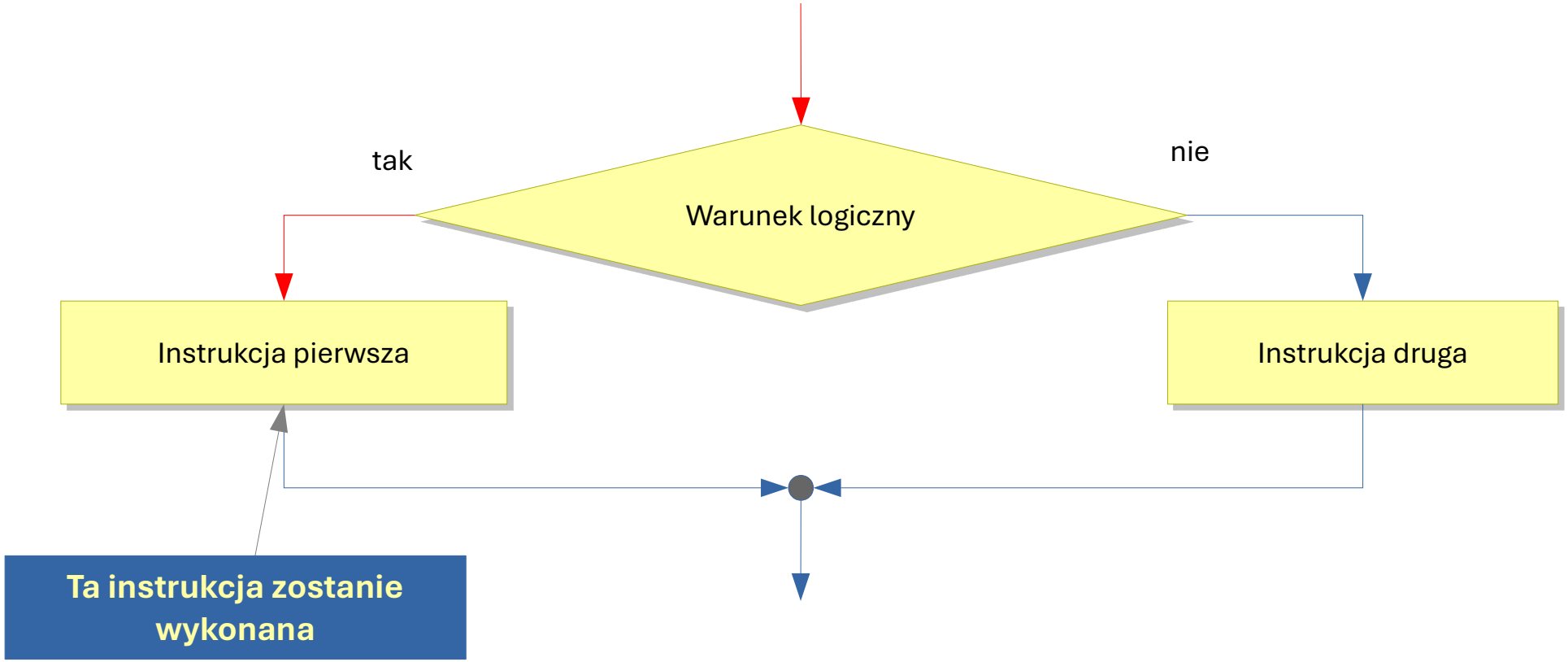
Instrukcja alternatywy – schemat blokowy



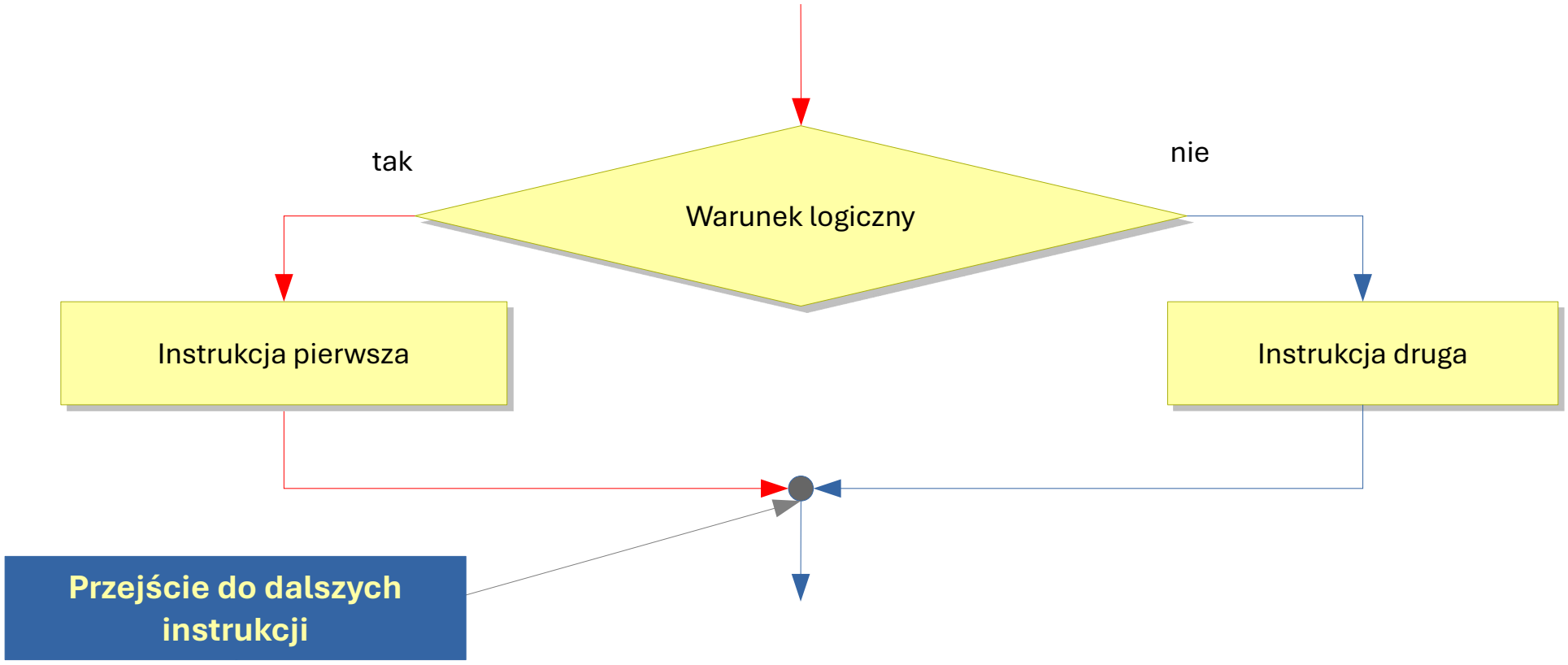
Instrukcja alternatywy – schemat blokowy



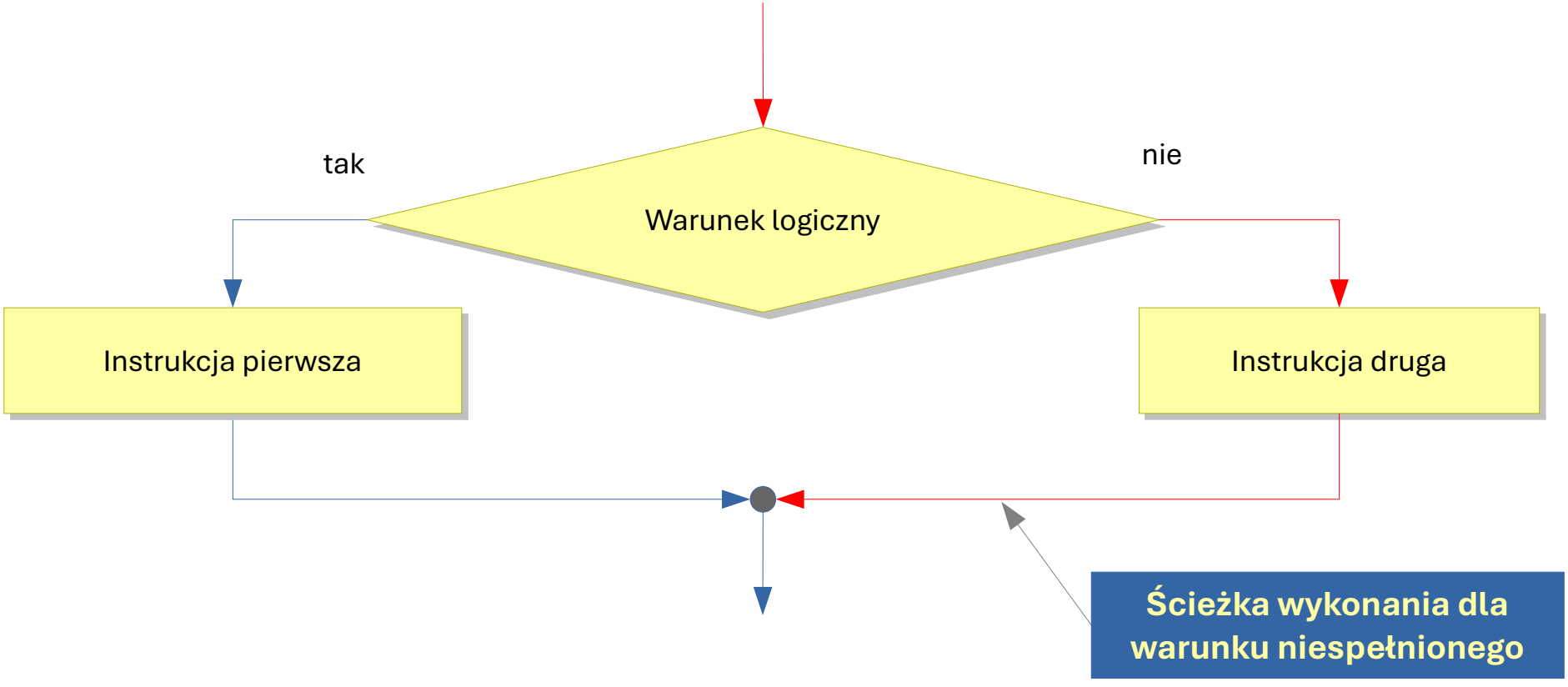
Instrukcja alternatywy – schemat blokowy



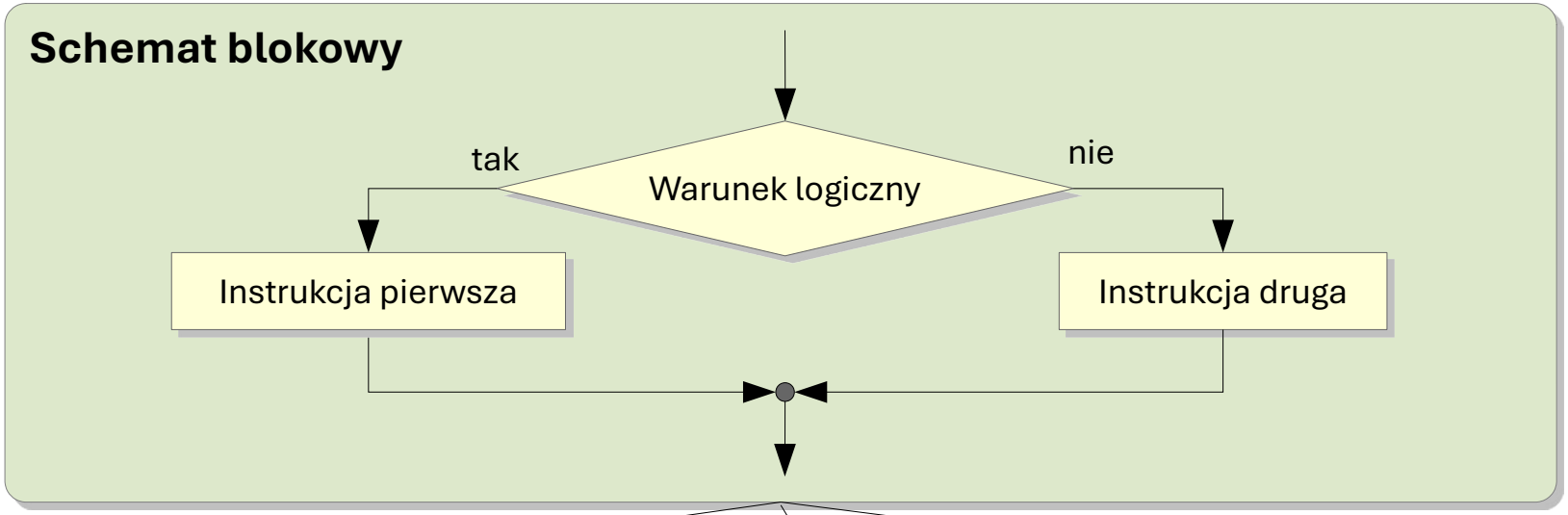
Instrukcja alternatywy – schemat blokowy



Instrukcja alternatywy – schemat blokowy



Instrukcja alternatywy – schemat blokowy




```
Jeżeli Warunek logiczny To
  Instrukcja pierwsza
W przeciwnym przypadku
  Instrukcja druga
```

Pseudokod

```
if Warunek logiczny:
  Instrukcja pierwsza
else:
  Instrukcja druga
```

Python

```
if(Warunek logiczny)
  Instrukcja pierwsza
else
  Instrukcja druga
```

C/C++/Java/C#

Python – zabezpieczenie przed ujemną wartością odległości

- Podejmowanie decyzji – użyjemy do tego **instrukcji alternatywy**.
- To *instrukcja sterująca przebiegiem wykonania programu*.

```
print("Przeliczam odległość wyrażoną w milach na kilometry")
mile = float(input("Podaj odległość w milach: "))
if mile >= 0:
    print("W kilometrach to:", mile * 1.6)
else:
    print("Odległość powinna być nieujemna")
```

Python – wcięcia przy edycji kodu są ważne

- Instrukcji w obrębie frazy **if** oraz **else** może być wiele.
- Instrukcje wewnętrzne muszą być zapisane z wcięciem.

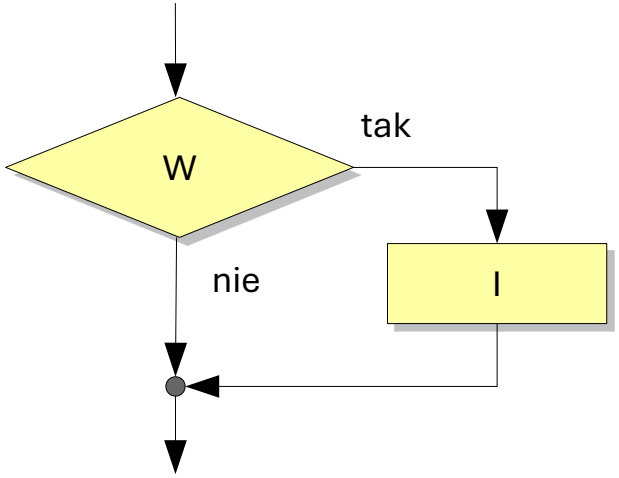
```
print("Przeliczam odległość wyrażoną w milach na kilometry")
mile = float(input("Podaj odległość w milach: "))
if mile >= 0:
    print("Wyświetlam wynik")
    print("W kilometrach to:", mile * 1.6)
else:
    print("Wystąpił błąd")
    print("Odległość powinna być nieujemna")
```

Instrukcja alternatywy i warunkowa

- Instrukcja alternatywy i warunkowa należą go grupy instrukcji sterujących wykonaniem programu. Wspólnie są nazywane *instrukcjami warunkowymi*.

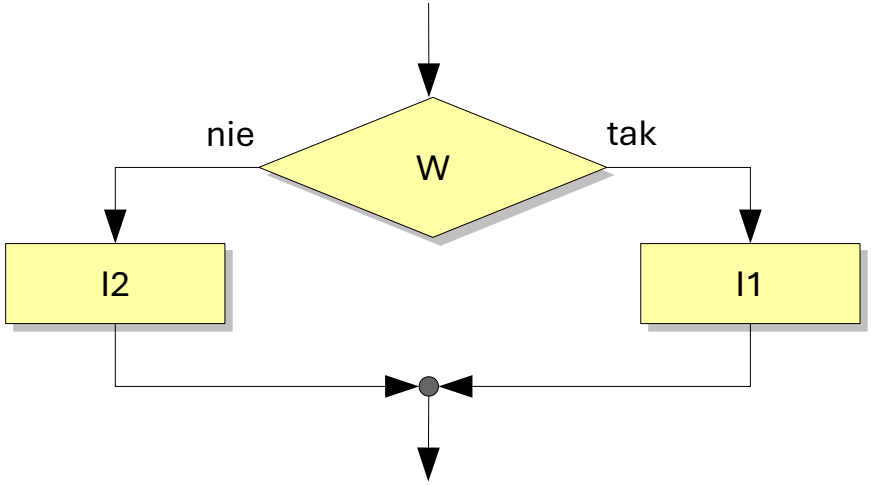
```
if W:
    I
```

Instrukcja warunkowa




```
if W:
    I1
else:
    I2
```

Instrukcja alternatywy



Python – zabezpieczenie przed nieprawidłową liczbą

- W przypadku wprowadzenia przez użytkownika ciągu znaków zawierającego znaki niedozwolone dla liczb, funkcja **float** nie będzie mogła dokonać konwersji.
- Spowoduje to wygenerowanie przez tę funkcję wyjątku *ValueError* i awaryjne przerwanie wykonania programu w przypadku nieobsłużenia tego wyjątku.

```
print("Przeliczam odległość wyrażoną w milach na kilometry")
mile = float(input("Podaj odległość w milach: "))
if mile >= 0:
    print("W kilometrach to:", mile * 1.6)
else:
    print("Odległość powinna być nieujemna")
```

Wyjątek: ValueError

Czy ktoś obsługuje ten wyjątek???

Python – zabezpieczenie przed nieprawidłową liczbą

```
print("Przeliczam odległość wyrażoną w milach na kilometry")
mile = float(input("Podaj odległość w milach: "))
if mile >= 0:
    print("W kilometrach to:", mile * 1.6)
else:
    print("Odległość powinna być nieujemna")
```

Wyjątek: ValueError

Wyjątek nie został obsłużony
Awaryjne zakończenie programu

```
Przeliczam odległość wyrażoną w milach na kilometry
Podaj odległość w milach: coś tam
Traceback (most recent call last):
  File "/home/main.py", line 2, in <module>
    mile = float(input("Podaj odległość w milach: "))
ValueError: could not convert string to float: 'coś tam'
```

Python – jak obsłużyć wyjątek?

```
try:
    print("Przeliczam odległość wyrażoną w milach na kilometry")
    mile = float(input("Podaj odległość w milach: "))
    if mile >= 0:
        print("W kilometrach to:", mile * 1.6)
    else:
        print("Odległość powinna być nieujemna")
except ValueError:
    print("Nieprawidłowa liczba")
```

Wyjątek: ValueError

```
Przeliczam odległość wyrażoną w milach na kilometry
Podaj odległość w milach: coś tam
Nieprawidłowa liczba
```

Python – jak obsłużyć wyjątek?

```
try:
    print("Przeliczam odległość wyrażoną w milach na kilometry")
    mile = float(input("Podaj odległość w milach: "))
    if mile >= 0:
```

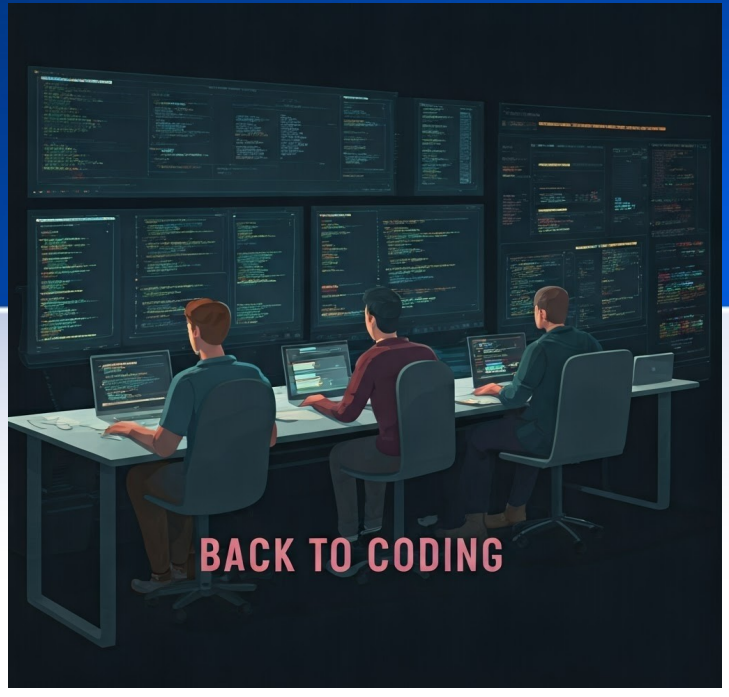
Uwaga – po obsłudze wyjątku nie możemy automatycznie wrócić do miejsca jego powstania i wznowić wykonanie przerwanych operacji – mleko się już rozlało...

```
Przeliczam odległość wyrażoną w milach na kilometry
Podaj odległość w milach: coś tam
Nieprawidłowa liczba
```

Python – instrukcje warunkowe można zagnieżdżać

```
try:
    print("Przeliczam odległość wyrażoną w milach na kilometry")
    mile = float(input("Podaj odległość w milach: "))
    if mile >= 0:
        if mile == 0:
            print("Po co liczyć dla 0?")
        else:
            print("W kilometrach to:", mile * 1.6)
    else:
        print("Odległość powinna być nieujemna")
except ValueError:
    print("Nieprawidłowa liczba")
```

C++

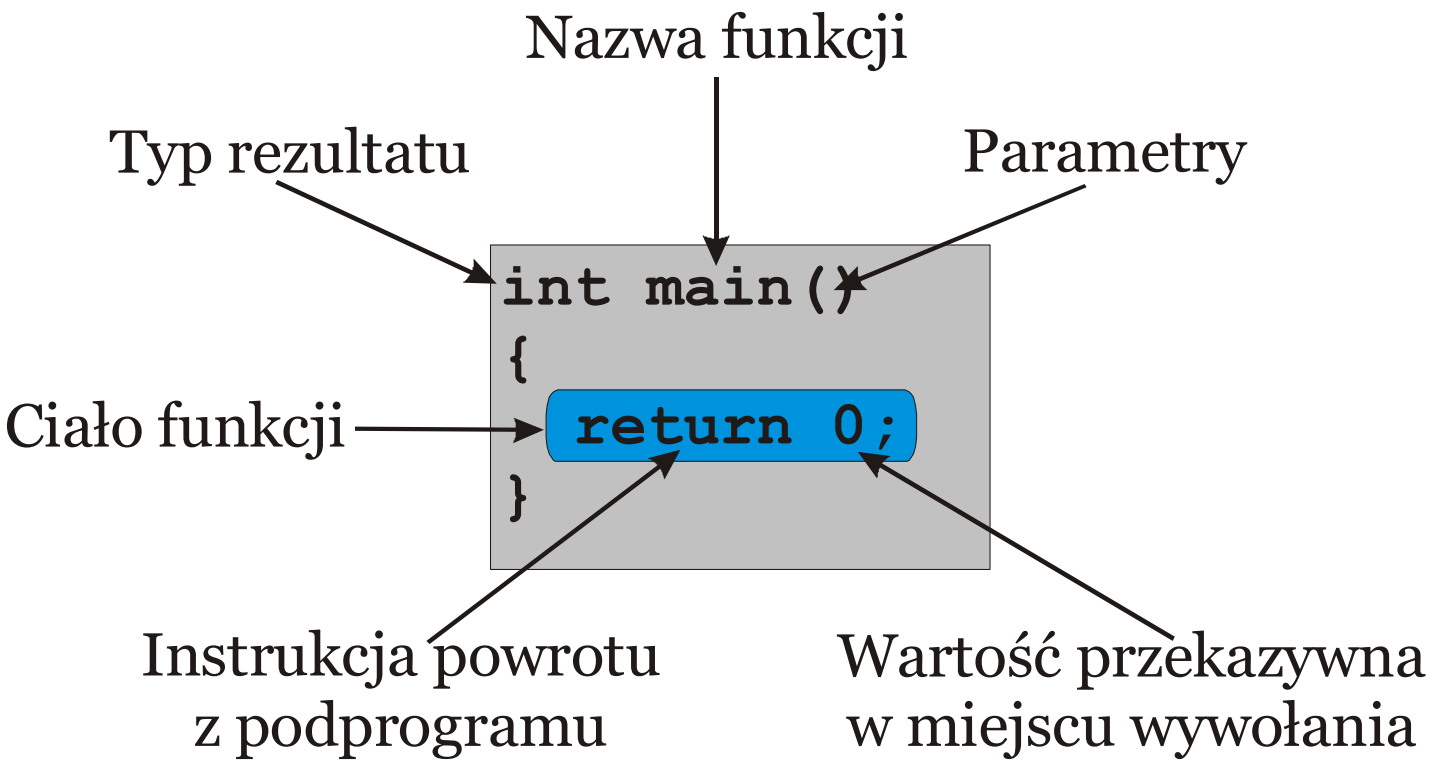


C++ – program, który robi nic

- Każdy program C/C++musi posiadać funkcję o nazwie **main**, stanowiącą tzw. punkt wejściowy programu.
- Od niej się rozpoczyna wykonanie programu napisanego w języku C/C++.

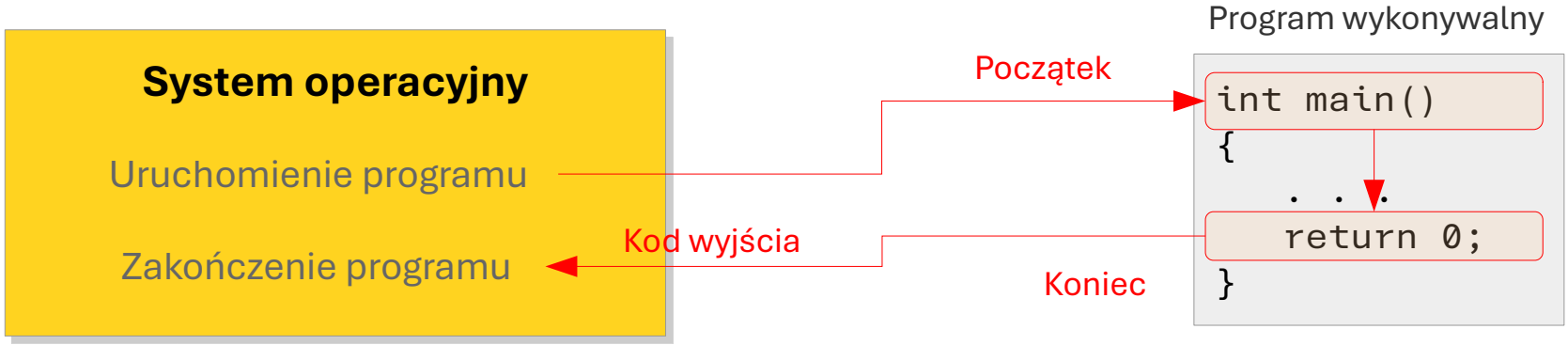
```
int main()
{
  return 0;
}
```

C++ – elementy definicji funkcji main



C++ – aby funkcja main zadziałała, należy ją wywołać

- Funkcja **main** stanowi punkt programu, od którego zaczyna się jego wywołanie.
- Upraszczając, można powiedzieć, że funkcja **main** jest wywoływana przez system operacyjny (nie zawsze jednak tak jest).
- Wartość, będąca rezultatem funkcji **main** jest przekazywana systemowi operacyjnemu, stanowiąc *kod wyjścia programu*.
- Kod wyjścia programu może być wykorzystywany w skryptach systemu operacyjnego oraz jako środek komunikacji pomiędzy procesami.



C++ – standaryzacja rezulatatu funkcji main

- Zamiast rezultatu w postaci bezwzględnych wartości można wykorzystać symbole:

  EXIT_SUCCESS — oznacza bezbłędne zakończenie programu.

  EXIT_FAILURE —zakończenie z informacją o błędzie.
- Stosowanie tych symboli jest zalecane przez standard POSIX. POSIX standaryzuje m.in. interfejs programistyczny, dotyczy głównie systemów klasy UNIX.
- Niestety, program zapisany tak się nie skompiluje:

```
int main()
{
  return EXIT_SUCCESS;
}
```

```
Message
In function `int main()':
`EXIT_SUCCESS' undeclared (first use this function)
(Each undeclared identifier is reported only once for each function it appears in.)
[Build Error] [main.o] Error 1
```

C++ – elementy spoza języka należy importować z jego bibliotek

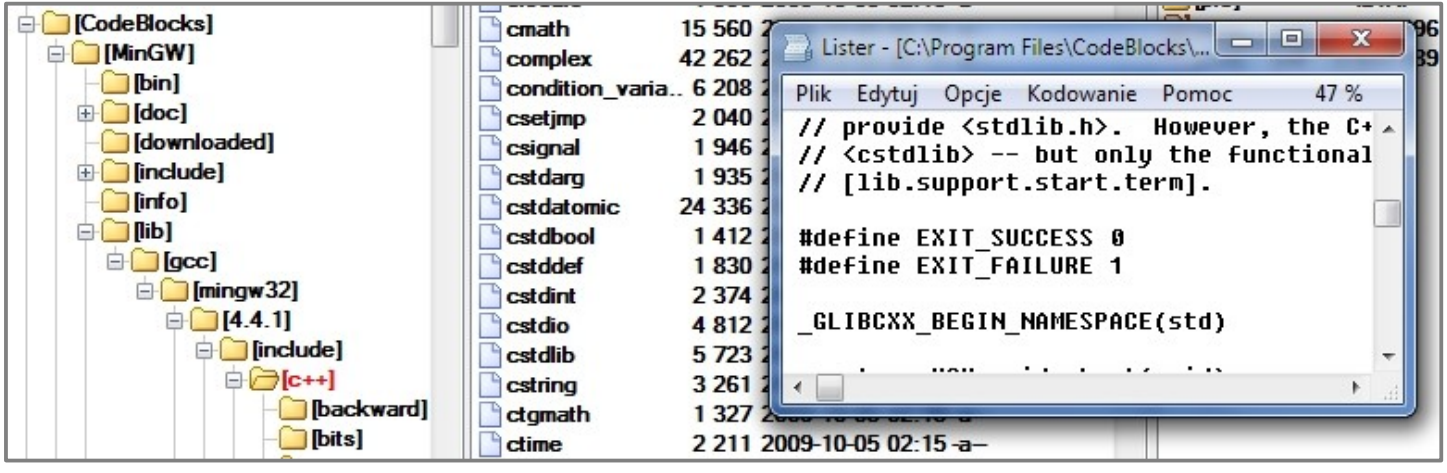
- Aby symbole: EXIT_SUCCESS i EXIT_FAILURE były dostępne w programie, należy włączyć do programu odpowiedni plik nagłówkowy.
- Pliki nagłówkowe zawierają „wizytówki” elementów zapisanych w bibliotekach języka C/C++.
- Dyrektywa **#include** powoduje włączenie do kodu źródłowego programu w języku C/C++ definicji i deklaracji niezbędnych dla kompilacji.

```
#include <cstdlib>
int main()
{
  return EXIT_SUCCESS;
}
```

Dyrektywa #include powoduje włączenie zawartości pliku *cstdlib*
Plik ten identyfikuje standardową bibliotekę języka C

C++ – dyrektywa #include

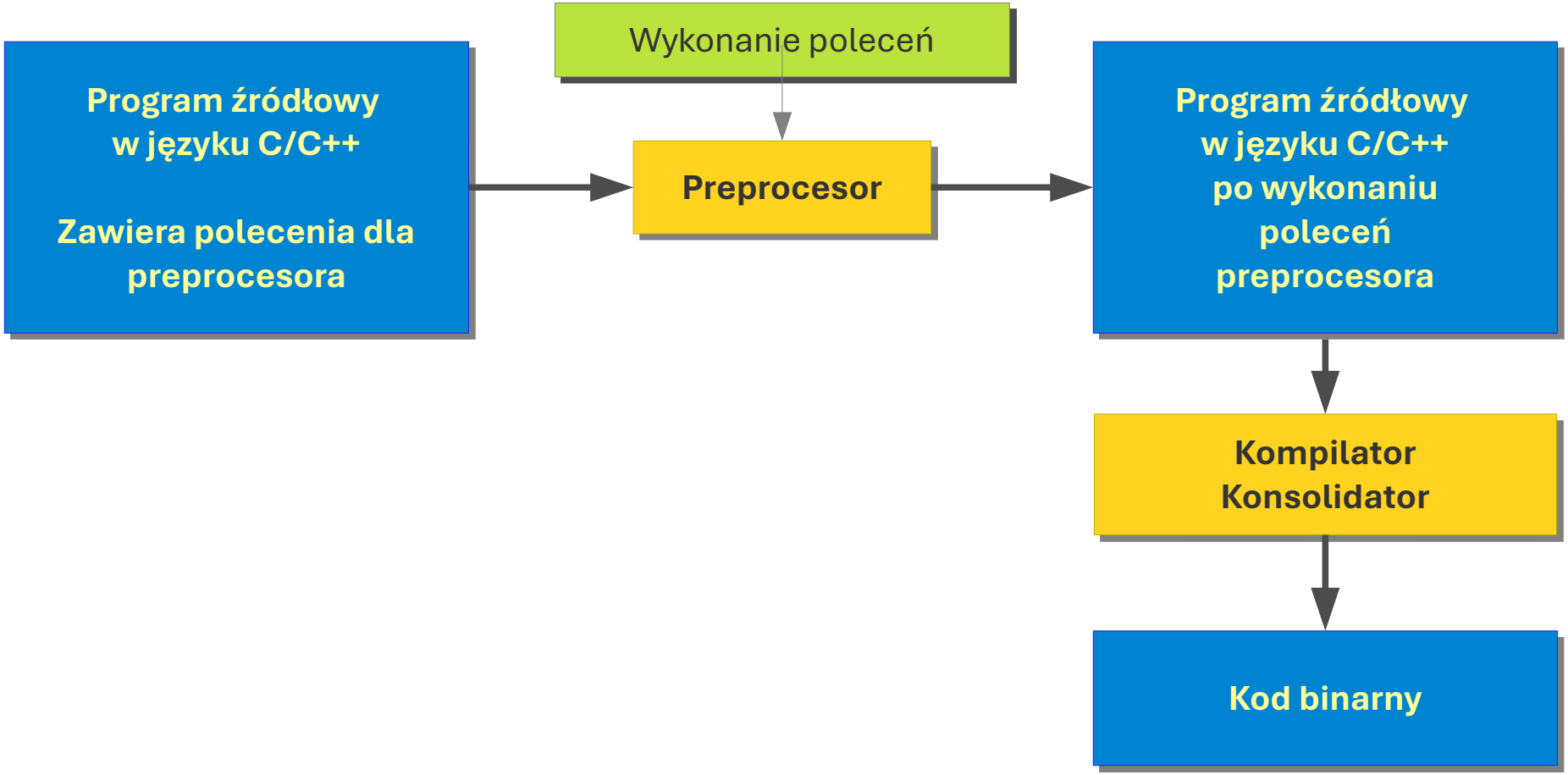
- Dyrektywa #include powoduje włączenie do kodu źródłowego definicji i deklaracji niezbędnych dla kompilacji.
- Informacje te są przechowywane w plikach nagłówkowych, są to pliki tekstowe. Znaleźć je można zwykle w katalogu include danego kompilatora.
- W pliku nagłówkowym cstdlib zdefiniowane są właśnie symbole EXIT_SUCCES oraz EXIT_FAILURE.



C/C++ – preprocesor

- Dyrektywa **#include** realizowana jest przez *preprocesor* języka C/C++.
- Preprocesor to zwykle osobny program, realizujący wstępne przetwarzanie kodu źródłowego programu w języku C (C++, Objective-C), realizowane przed przekazaniem programu na wejście kompilatora.
- Zadaniem preprocesora jest *wyszukanie* w tekście źródłowym programu przeznaczonych dla niego poleceń, oraz ich *wykonanie*.
- Wykonanie polecenia oznacza zwykle operację na tekście źródłowym — zamianę jednego tekstu na inny, włączenie zawartości jakiegoś innego pliku, pominięcie pewnego fragmentu tekstu itp.

C/C++ – preprocesor, kolejność działań w trakcie kompilacji



C/C++ – preprocesor, działanie

```
#include <cstdlib>

int main()
{
  return EXIT_SUCCESS;
}
```

Dyrektywa #include powoduje włączenie zawartości całego pliku c*stdlib*, w tym definicje symboli *EXIT_SUCCESS* i *EXIT_FAILURE*

```
. . .
#define EXIT_SUCCESS 0
#define EXIT_FAILURE 1
. . .

int main()
{
  return EXIT_SUCCESS;
}
```

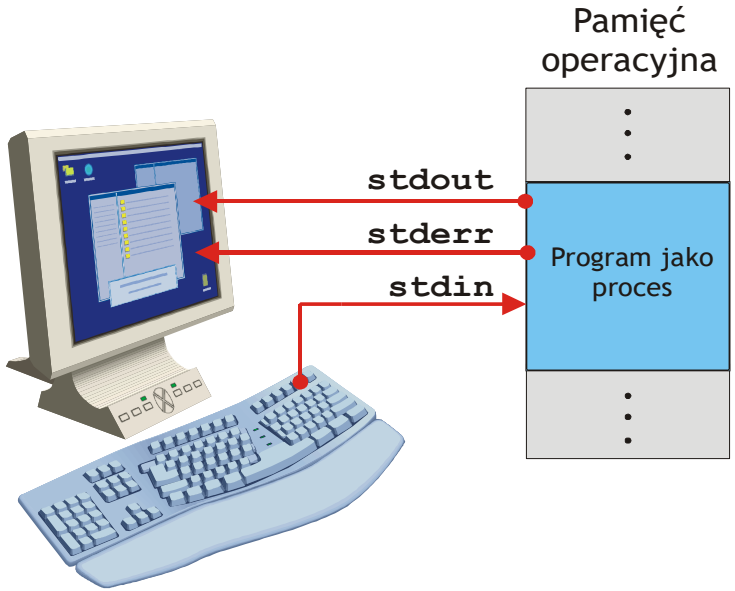
Dyrektywa #define to także polecenie preprocesora, jej znaczenie omówione zostanie później

Standardowe strumienie programu w języku C

- *stdin* — standardowy strumień wejściowy programu, jest zwykle skojarzony z klawiaturą;
- *stdout* — standardowy strumień wyjściowy programu, jest zwykle skojarzony z ekranem monitora;
- *stderr* — standardowy strumień wyjściowy błędów programu, jest zwykle skojarzony również z ekranem monitora.

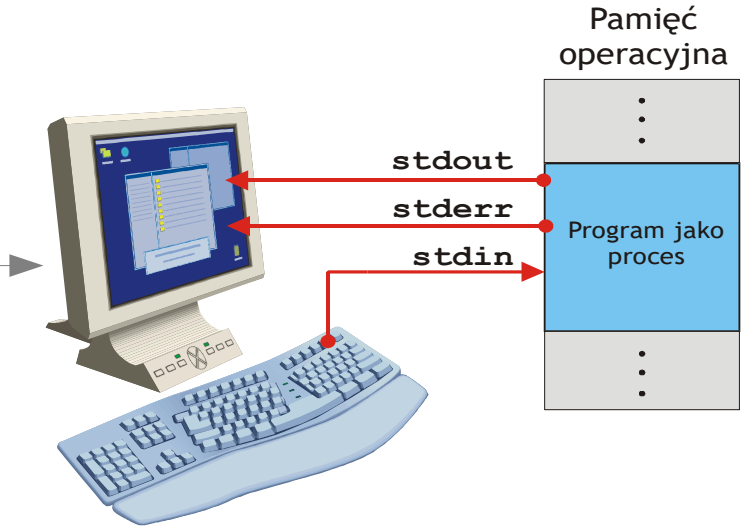
Strumienie *stdin*, *stdout* reprezentują normalny kanał komunikacji programu z użytkownikiem

Strumień *stderr* zarezerwowany jest do wyświetlania komunikatów diagnostycznych programu



Standardowe strumienie programu w języku C

Standardowe skojarzenie strumieni



Redyrekcja (przekierowanie) strumieni z poziomu systemu operacyjnego:

```
C:\>test.exe > wyjscie.txt
```

```
C:\>test.exe < wejscie.txt
```

```
C:\>test.exe < wejscie.txt > wyjscie.txt
```

Standardowe strumienie programu w języku C++

W języku C++ rolę strumieni We-Wy przejmują predefiniowane obiekty klas **istream** oraz **ostream**.

Aby skorzystać z obiektów *cin* i cout należy włączyć plik nagłówkowy iostream:

```
#include <iostream>
```

- **cin** — strumień reprezentujący standardowe wejście programu, odpowiada strumieniowi stdin z C. Strumień cin odczytuje dane i zapisuje je do odpowiednich zmiennych.
- **cout** — strumień reprezentujący standardowe wyjście programu. Odpowiada strumieniowi stdout z C.
- **cerr** — niebuforowany strumień wyjściowy błędów. Odpowiada strumieniowi stderr z C.
- **clog** — buforowany strumień wyjściowy błędów. Odpowiada strumieniowi stderr z C.

Wracamy do programu przeliczania odległości...

```
#include <cstdlib>
#include <iostream>

using namespace std;

int main()
{
  return EXIT_SUCCESS;
}
```

Dyrektywa #include powoduje włączenie zawartości pliku *iostream*, zawierającego definicje niezbędne do obsługi standardowego wejścia-wyjścia w języku C++

Wracamy do programu przeliczania odległości...

```
#include <cstdlib>
#include <iostream>

using namespace std;

int main()
{
  return EXIT_SUCCESS;
}
```

W języku C++ wprowadzono przestrzenie nazw. Pozwalają one na powtarzanie tych samych nazw w różnych przestrzeniach. Ta linia programu mówi, że będziemy korzystali ze standardowej przestrzeni nazw – std.

Wracamy do programu przeliczania odległości...

```
#include <cstdlib>
#include <iostream>

using namespace std;

int main()
{
  cout << "Przeliczam odległość wyrażoną w milach na kilometry" << endl;

  return EXIT_SUCCESS;
}
```

Wyprowadzanie danych odbywa się z wykorzystaniem operatora <<, zwanego *wstawiaczem* (ang. inserter). Dane są wyprowadzane zgodnie z ich typem, istnieje możliwość formatowania postaci wyjściowej.

Manipulator **endl** powodują opróżnienie bufora strumienia **cout**, wstawia znak nowej linii przed opróżnieniem bufora.

Sekwencje specjalne w literałach znakowych i łańcuchowych

- W językach C i C++ można używać specjalnych sekwencji znakowych. Niektóre z nich działają również w innych językach, w tym w języku Python.
- Dodatkowo sekwencje specjalne są wykorzystywane do zapisu pewnych „niewygodnych” stałych znakowych.

Sekwencja	Wartość	Znak	Znaczenie
\a	0x07	BEL	Audible bell
\b	0x08	BS	Backspace
\f	0x0C	FF	Formfeed
\n	0x0A	LF	Newline (linefeed)
\r	0x0D	CR	Carriage return
\t	0x09	HT	Tab (horizontal)
\v	0x0B	VT	Vertical tab
\\	0x5c	\	Backslash
\'	0x27	'	Apostrof
\"	0x22	"	Cudzysłów
\?	0x3F	?	Pytajnik
\O		any	O = łańcuch ósemkowych cyfr
\xH		any	H = łańcuch szesnastkowych cyfr
\XH		any	H = łańcuch szesnastkowych cyfr

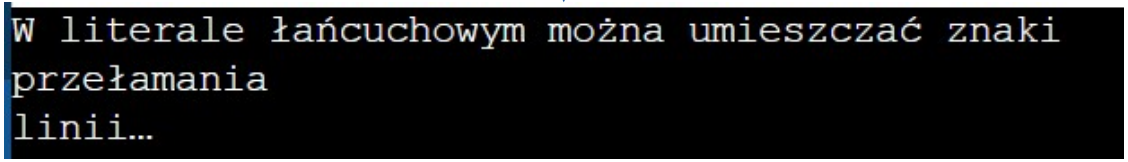
Literały znakowe i łańcuchowe

- W C/C++ literały łańcuchowe zapisujemy tylko w pomiędzy symbolami cudzysłowów.
- Symbole apostrofów służą do zapisywania literałów znakowych.

"Łańcuchy znaków żargonowo nazywa się \”stringami\”..."

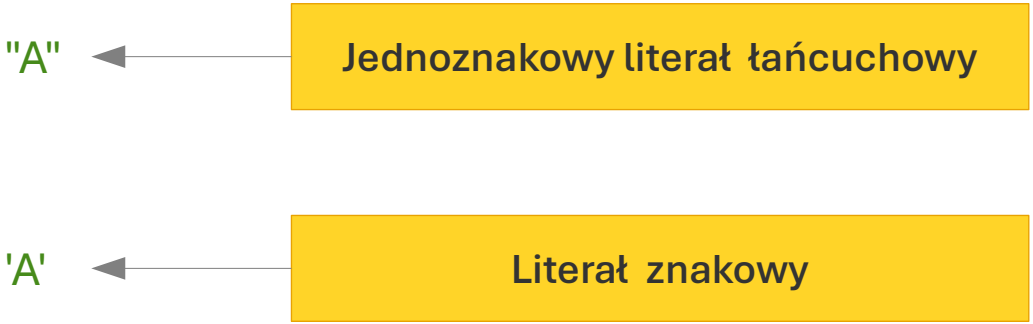
"W literale łańcuchowym można umieszczać znaki\nprzełamania\nlinii..."

```
cout << "W literale łańcuchowym można umieszczać znaki\nprzełamania\nlinii…"
```



Literały znakowe i łańcuchowe

- Jednoznakowy literał łańcuchowy to nie to samo, co literał znakowy, ich wewnętrzne reprezentacje nie są takie same.



Sekwencje specjalne w literałach znakowych i łańcuchowych

- Zamiast manipulatora **endl** można używać znaku **\n**.
- Manipulator **endl** powoduje opróżnienie bufora strumienia wyjściowego i przejście do nowej linii, sekwencja **\n** powoduje tylko przejście do nowej linii.
- Sekwencje sterujące mogą występować jako literały znakowe lub być częścią literałów łańcuchowych.

```
'a'     '*'     '\n'        '\t'        '\a'        '\\'        '\''        '\”'
```

```
”Język C++”     "c:\\system\\trash"     ”\nRaz\nDwa\nTrzy”     ”\”Cudzysłów\””
```

Do reprezentowania znaków w językach C/C++ służy typ znakowy *char*

C++ – deklaracja zmiennej

```
#include <cstdlib>
#include <iostream>

using namespace std;

int main()
{
    float mile;

    cout << "Przeliczam odległość wyrażoną w milach na kilometry" << endl;

    return EXIT_SUCCESS;
}
```

Jawna deklaracja zmiennej mile, zawierająca określenie jej typu

C++ – wyprowadzenie zachęty do wpisania odległości

```
#include <cstdlib>
#include <iostream>

using namespace std;

int main()
{
    float mile;

    cout << "Przeliczam odległość wyrażoną w milach na kilometry" << endl;
    cout << "Podaj odległość w milach: ";

    return EXIT_SUCCESS;
}
```

C++ – wczytanie odległości z konwersją

```
#include <cstdlib>
#include <iostream>

using namespace std;

int main()
{
    float mile;

    cout << "Przeliczam odległość wyrażoną w milach na kilometry" << endl;
    cout << "Podaj odległość w milach: ";

    cin >> mile;

    return EXIT_SUCCESS;
}
```

Wczytanie liczby typu *float* ze strumienia wejściowego programu.
Dokonywana jest automatyczna konwersja wartości – o ile to możliwe
Uwaga – znak końca linii wstawiany poprzez naciśnięcie klawisza Enter
Może pozostać w buforze klawiatury, co może spowodować problemy
przy odczycie następnych informacji z *cin*

C++ – wczytanie odległości z konwersją – dokładniej

```
#include <cstdlib>
#include <iostream>

using namespace std;

int main()
{
    float mile;

    cout << "Przeliczam odległość wyrażoną w milach na kilometry" << endl;
    cout << "Podaj odległość w milach: ";

    cin >> mile;

    return EXIT_SUCCESS;
}
```

cin — globalny obiekt *cin* umożliwia wczytywanie danych z wykorzystaniem operatora >> zwanego wydobywaczem (ang. *extractor*), który przeprowadza konieczne konwersje w trakcie pobierania danych. Operator >> pomija wiodące „białe” znaki w strumieniu wejściowym.

C++ – wyprowadzenie wyniku

```
#include <cstdlib>
#include <iostream>

using namespace std;

int main()
{
    float mile;

    cout << "Przeliczam odległość wyrażoną w milach na kilometry" << endl;
    cout << "Podaj odległość w milach: ";

    cin >> mile;

    cout << "W kilometrach to: " << mile * 1.6 << endl;

    return EXIT_SUCCESS;
}
```

C++ – zabezpiecznie przed nieprawidłową liczbą, wersja I

```
. . .

    float mile;

    cout << "Przeliczam odległość wyrażoną w milach na kilometry" << endl;
    cout << "Podaj odległość w milach: ";
    cin >> mile;

    if(cin.fail())
       cout << "Błędna liczba" << endl;
    else
       cout << "W kilometrach to: " << mile * 1.6 << endl;

. . .
```

Rezultatem *cin.fail()* jest wartość *true* jeżeli ostatnia operacja wejścia zakończyła się niepowodzeniem, wartość *false* w przeciwnym przypadku.

C++ – zabezpiecznie przed nieprawidłową liczbą, wersja II

```
string mileJakoTekst;
float mile;

cout << "Przeliczam odległość wyrażoną w milach na kilometry" << endl;
cout << "Podaj odległość w milach: ";
cin >> mileJakoTekst;
try
{
    mile = stof(mileJakoTekst);
    cout << "W kilometrach to: " << mile * 1.6 << endl;
}
catch(...)
{
     cout << "Błędna liczba" << endl;
}
```

Ponieważ zapis parametrów catch w C++ jest nieintuicyjny, użycie ... zapewnia obsługę wszystkich wyjątków

C++ – zabezpiecznie przed ujemną wartością odległości

```
string mileJakoTekst;
float mile;

cout << "Przeliczam odległość wyrażoną w milach na kilometry" << endl;
cout << "Podaj odległość w milach: ";
cin >> mileJakoTekst;
try
{
    mile = stof(mileJakoTekst);
    if(mile >= 0)
       cout << "W kilometrach to: " << mile * 1.6 << endl;
    else
       cout << "Odległość powinna być nieujemna” << endl;
}
catch(...)
{
     cout << "Błędna liczba" << endl;
}
```

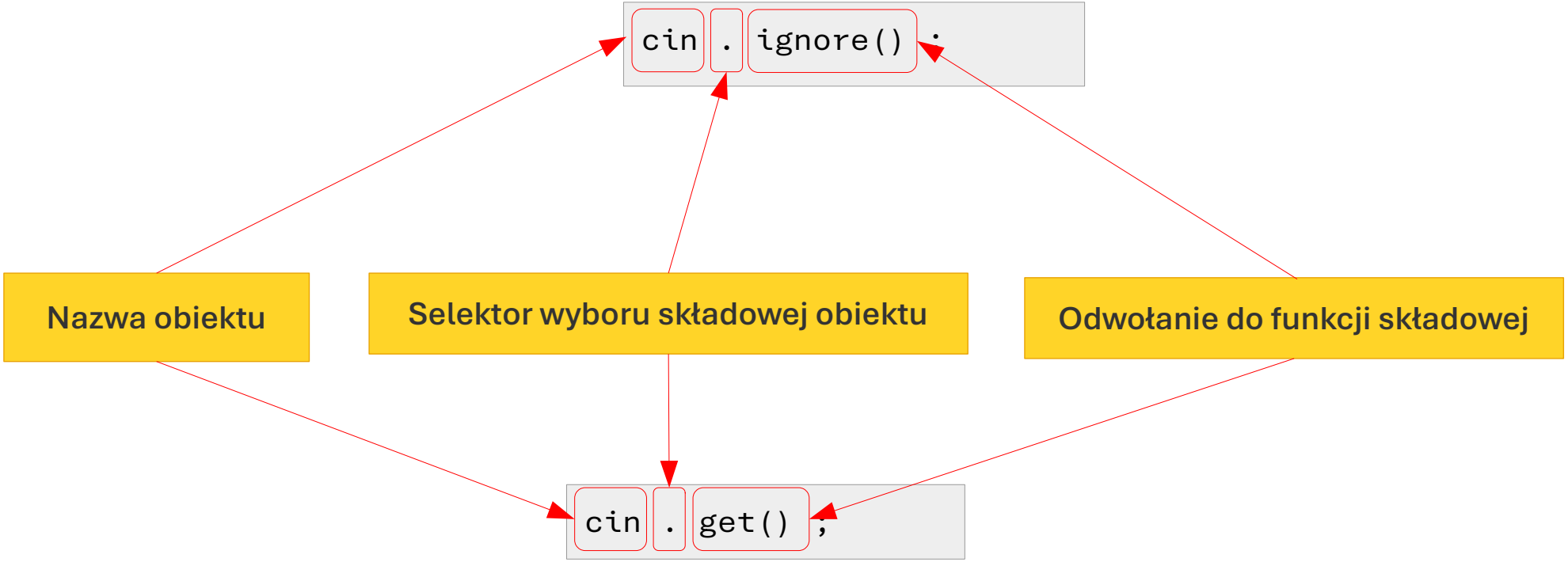
C++ – gdy program uruchamia się w zewnętrznej konsoli

```
#include <cstdlib>
#include <iostream>

using namespace std;

int main()
{
  . . .
  . . .
  cout << endl << "Nacisnij Enter by zakonczyc program...";
  cin.ignore();
  cin.get();

  return EXIT_SUCCESS;
}
```

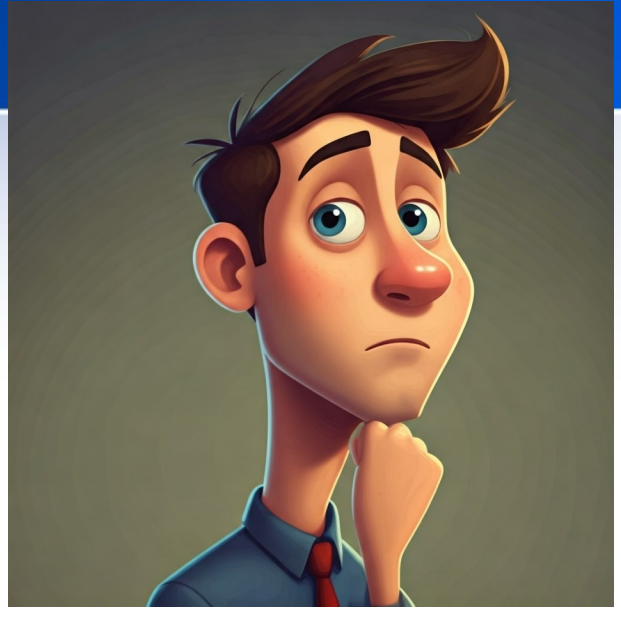
cin.ignore() pomija znak Enter pozostały w buforze po wprowadzeniu odległości.

cin.get() pobiera znak ze strumienia wejściowego.

C++ – cin to tak naprawdę obiekt



Sprawa do przemyślenia – szablon dla programu przeliczania wartości



Szablon dla programu przeliczania wartości – parametryzacja

```
przywitanie = "Przeliczam odległość wyrażoną w milach na kilometry"
pytanie_o_wartosc = "Podaj odległość w milach: "
komunikat_po_przeliczeniu = "W kilometrach to:"
komunikat_gdy_niepoprawny_zakres = "Odległość powinna być nieujemna"
komunikat_gdy_nieprawidlowa_liczba = "Nieprawidłowa liczba"
przelicznik = 1.6
try:
    print(przywitanie)
    wartosc_do_przeliczenia = float(input(pytanie_o_wartosc))
    if wartosc_do_przeliczenia >= 0:
        print(komunikat_po_przeliczeniu, wartosc_do_przeliczenia * przelicznik)
    else:
        print(komunikat_gdy_niepoprawny_zakres)
except ValueError:
    print(komunikat_gdy_nieprawidlowa_liczba)
```

Szablon dla programu przeliczania wartości

```
przywitanie = "Przeliczam odległość wyrażona w milach na kilometry"
pytanie_o_wartosc = "Poc
komunikat_po_przeliczeniu
komunikat_gdy_niepoprawny_zakres = "Odległość powinna być nieujemna"
komunikat_gdy_nieprawidlowa_liczba = "Nieprawidłowa liczba"
przelicznik = 1.6
try:
    print(przywitanie)
    wartosc_do_przeliczenia = float(input(pytanie_o_wartosc))
    if wartosc_do_przeliczenia >= 0:
        print(komunikat_po_przeliczeniu, wartosc_do_przeliczenia * przelicznik)
    else:
        print(komunikat_gdy_niepoprawny_zakres)
except ValueError:
    print(komunikat_gdy_nieprawidlowa_liczba)
```

To także można sparametryzować

Szablon dla programu przeliczania wartości

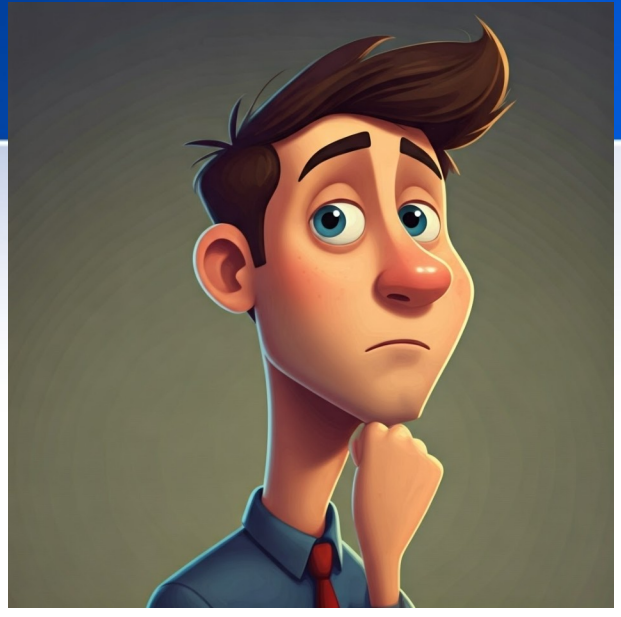
```
przywitanie = "Przeliczam odległość wyrażoną w milach na kilometry"
pytanie_o_wartosc = "Podaj odległość w milach: "
komunikat_po_przeliczeniu = "W kilometrach to:"
komunikat_gdy_niepoprawny_zakres = "Odległość powinna być nieujemna"
komunikat_gdy_nieprawidlowa_liczba = "Nieprawidłowa liczba"
przelicznik = 1.6
ograniczenie_dolne = 0
ograniczenie_gorne = math.inf
try:
    print(przywitanie)
    wartosc_do_przeliczenia = float(input(pytanie_o_wartosc))
    if wartosc_do_przeliczenia >= ograniczenie_dolne \
    and wartosc_do_przeliczenia <= ograniczenie_gorne:
        print(komunikat_po_przeliczeniu, wartosc_do_przeliczenia * przelicznik)
    else:
        print(komunikat_gdy_niepoprawny_zakres)
except ValueError:
    print(komunikat_gdy_nieprawidlowa_liczba)
```

Szablon dla programu przeliczania wartości

```
przywitanie = "Przeliczam odległość wyrażoną w milach na kilometry"
pytanie_o_wartosc = "Podaj odległość w milach: "
komunikat_po_przeliczeniu = "W kilometrach to:"
komunikat_gdy_niepoprawny_zakres = "Odległość powinna być nieujemna"
komunikat_gdy_nieprawidlowa_liczba = "Nieprawidłowa liczba"
przelicznik
ograniczenie
ograniczenie
try:
    print(p
    wartosc_
    if wartosc_do_przeliczenia >= ograniczenie_dolne
    and wartosc_do_przeliczenia <= ograniczenie_gorne:
        print(komunikat_po_przeliczeniu, wartosc_do_przeliczenia * przelicznik)
    else:
        print(komunikat_gdy_niepoprawny_zakres)
except ValueError:
    print(komunikat_gdy_nieprawidlowa_liczba)
```

Czy warto tak kombinować?

A co gdy chcę wiedzieć więcej?



Suplement – dodatkowe informacje

O formatowaniu łańcuchów znaków w języku Python:

https://zetcode.com/python/fstring/

https://www.w3schools.com/python/python_strings_format.asp

Bardziej zaawansowany opis formatowania łańcuchów znaków w języku Python:

https://www.w3schools.com/python/python_string_formatting.asp

O try-except w języku Python:

https://www.w3schools.com/python/python_try_except.asp

https://python.land/deep-dives/python-try-except

Dziękuję za uwagę

roman.siminski@us.edu.pl

roman@siminskionline.pl