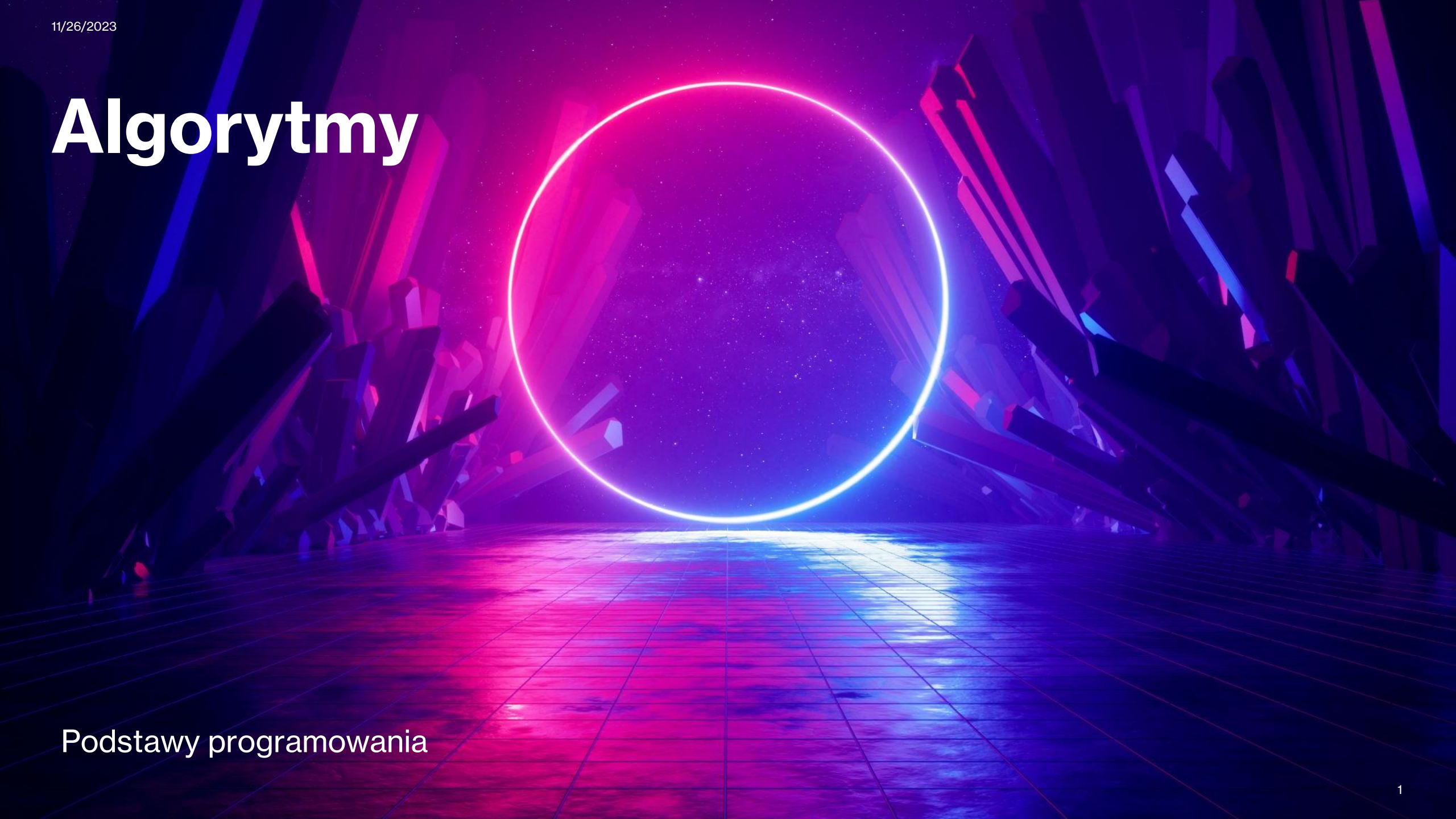
Algorytmy



Podstawy programowania

Zanim zaczniemy, parę definicji

- **Zmienna**
  symboliczna nazwa, która przechowuje wartość. W programowaniu, zmienne są używane do przechowywania danych, które mogą być modyfikowane i używane w trakcie działania programu.
- **Instrukcja warunkowa**
  Instrukcja warunkowa to konstrukcja w programowaniu, która pozwala na wykonanie różnych działań w zależności od spełnienia określonego warunku.
- **Rekurencja**
  Sortowanie to proces uporządkowania elementów w określonej kolejności, na przykład rosnąco lub malejąco. Istnieje wiele algorytmów sortowania, takich jak sortowanie bąbelkowe, sortowanie przez wstawianie i sortowanie szybkie.

Zanim zaczniemy, parę definicji

- **Wyszukiwanie**

  Wyszukiwanie oznacza znalezienie określonego elementu w zbiorze danych. Istnieją różne algorytmy wyszukiwania, takie jak wyszukiwanie liniowe i wyszukiwanie binarne.

- **Tablica**

  Tablica (lub lista) to struktura danych, która pozwala przechowywać więcej niż jedną wartość w jednej zmiennej. Elementy tablicy są przechowywane na podstawie indeksów.

- **Złożoność czasowa i przestrzenna**

  Złożoność czasowa odnosi się do ilości czasu potrzebnego do wykonania algorytmu, podczas gdy złożoność przestrzenna odnosi się do ilości pamięci potrzebnej do wykonania algorytmu. Algorytmy są często analizowane pod kątem ich złożoności czasowej i przestrzennej.

Czym są te algorytmy?

- Po krótce algorytm to zestaw instrukcji, według których nasz program komputerowy wykonuje określone zadanie krok po kroku jak zapisaliśmy w naszej instrukcji.

Ciekawostka

Słowo „algorytm” pochodzi od łacińskiego słowa algorithmus, oznaczającego wykonywanie działań przy pomocy liczb arabskich

Gdzie znajdziemy algorytmy?

- Poza naszymi programami komputerowymi możemy znaleźć algorytmy w życiu codziennym, jak np. w przepisach kulinarnych, gdzie mamy opisane krok po kroku, co należy wykonywać, aby jedzenie wyszło poprawnie, czy instrukcja składania mebli, aby poprawnie zbudować dany mebel.

Przykład algorytmu obliczającego sumę kolejnych liczb

- Opis: Zbuduj algorytm obliczający sumę 5 kolejnych liczb naturalnych
- Działanie algorytmu

1. Rozpocznij program
2. Utwórz zmienną **suma**
3. Dopóki użytkownik nie poda pięciu zmiennych
   Dodawaj podane liczby do zmiennej **suma**
4. Po zakończeniu pętli wyświetl wartość **suma**

Klasyfikacje algorytmów

- Mamy tak naprawdę wiele różnych sposobów podziału algorytmów, ale najważniejszymi z nich są m.in.:
  + Algorytmy sekwencyjne
  + Algorytmy warunkowe
  + Algorytmy iteracyjne (pętle)
  + Algorytmy rekurencyjne

Algorytmy sekwencyjne

- Algorytmy sekwencyjne to zestawy kroków, które są wykonywane po kolei, jeden po drugim, bez żadnych powtórzeń ani skoków. Są to proste instrukcje wykonywane w określonej kolejności, podobne do przepisu kulinarnej.
- *Przykład:* Przepis na herbatę: gotowanie wody, zaparzanie herbaty, dodawanie cukru itp.

Algorytmy warunkowe

- Algorytmy warunkowe zawierają instrukcje, które są wykonywane tylko wtedy, gdy określony warunek jest spełniony. Jeśli warunek jest prawdziwy, wykonuje się jedna sekwencja kroków; jeśli jest fałszywy, wykonuje się inna sekwencja kroków.
- *Przykład:* Decyzja czy zabrać parasol - jeśli pada deszcz, zabierz parasol, w przeciwnym razie nie.

Algorytmy iteracyjne (pętle)

- Algorytmy iteracyjne, czyli pętle, polegają na powtarzaniu określonych kroków przez określoną liczbę razy lub do momentu spełnienia pewnego warunku. Pozwalają na automatyczne powtarzanie pewnych kroków w algorytmie.
- *Przykład:* Zliczanie od 1 do 10 - użyj pętli, aby zaczynać od 1 i dodawaj 1 do liczby na każdej iteracji, aż do osiągnięcia 10.

Algorytmy rekurencyjne

- Algorytmy rekurencyjne to takie, które wywołują same siebie w celu rozwiązania problemu. Jest to jak funkcja, która używa swojej własnej definicji w procesie rozwiązywania problemu.
- *Przykład:* Obliczanie silni - silnia n! jest równa n * (n-1)!; zdefiniuj funkcję silnia(n), która wywołuje samą siebie dla n-1, aż do osiągnięcia podstawowego przypadku, którym jest silnia(1) = 1.

Najpopularniejsze algorytmy stosowane w programowaniu

- **Sortowanie Bąbelkowe:**
  Algorytm sortowania, który porównuje sąsiednie elementy i zamienia je, jeśli są w niewłaściwej kolejności. Proces ten powtarza się dla każdej pary elementów aż do momentu, gdy cała lista zostanie posortowana.
- **Sortowanie Przez Wstawianie:**
  Algorytm sortowania, który przechodzi przez listę od początku do końca i wstawia każdy element na odpowiednie miejsce w posortowanej części listy
- **Wyszukiwanie Liniowe**
  Prosty algorytm wyszukiwania, który przegląda listę elementów jeden po drugim, aby znaleźć określony element. Jest prosty, ale ma złożoność czasową O(n), co oznacza, że czas wyszukiwania rośnie wraz z liczbą elementów w liście.

Najpopularniejsze algorytmy stosowane w programowaniu

- **Algorytm Euklidesa (Największy Wspólny Dzielnik):**
  Algorytm matematyczny do znajdowania największego wspólnego dzielnika dwóch liczb całkowitych. Opiera się na prostym pomyśle dzielenia jednej liczby przez drugą i zastępowania większej liczby mniejszą.
- **Wyszukiwanie Binarne**
  Algorytm wyszukiwania, który działa na posortowanej liście. Porównuje elementy z elementem, który chcemy znaleźć, i zmniejsza obszar poszukiwań na pół za każdym razem, aż odnajdzie odpowiedni element

Złożoność czasowa w algorytmie

- **Złożoność czasowa O(1) (stała)**

  Algorytm ma stałą złożoność czasową, co oznacza, że czas wykonania nie zależy od rozmiaru danych wejściowych. Przykładem jest dostęp do konkretnego elementu w tablicy

- **Złożoność czasowa O(n) (liniowa)**

  Algorytm ma liniową złożoność czasową, co oznacza, że czas wykonania rośnie proporcjonalnie do rozmiaru danych wejściowych. Przykładem jest przeszukiwanie listy w poszukiwaniu określonego elementu w przypadku wyszukiwania liniowego.

Złożoność czasowa w algorytmie

- **Złożoność czasowa O(n^2) (kwadratowa):**
  Algorytm ma złożoność kwadratową, co oznacza, że czas wykonania rośnie proporcjonalnie do kwadratu rozmiaru danych wejściowych. Przykładem jest algorytm sortowania bąbelkowego.
- **Złożoność czasowa O(log n) (logarytmiczna)**
  Algorytm ma logarytmiczną złożoność czasową, co oznacza, że czas wykonania rośnie w sposób logarytmiczny w stosunku do rozmiaru danych wejściowych. Przykładem jest algorytm wyszukiwania binarnego w posortowanej liście.

Złożoność czasowa w algorytmie

- **Złożoność czasowa O(n log n) (liniowo-logarytmiczna):**

  Algorytm ma liniowo-logarytmiczną złożoność czasową, która jest efektywniejsza niż czysta kwadratowa, ale mniej efektywna niż liniowa lub logarytmiczna. Przykładem jest algorytm sortowania szybkiego

- **Złożoność czasowa O(2^n) (wykładnicza)**

  Algorytm ma złożoność wykładniczą, co oznacza, że czas wykonania rośnie eksponencjalnie w stosunku do rozmiaru danych wejściowych. Algorytmy o wykładniczej złożoności czasowej są bardzo wolne dla większych danych wejściowych. Przykładem jest algorytm rozwiązywania problemu plecakowego metodą siłową.

Jak zapisujemy algorytm?

- Algorytm możemy zapisać w postaci:
  + Opisu słownego
  + Listy kroków
  + Schematu blokowego
  + Pseudokodu / Kodu w danym języku

Przykładowa specyfikacja problemu algorytmicznego

- Przykład:
  + Wyznacz oraz wyświetl największą liczbę z trzech podanych liczb całkowitych, naturalnych

Opis słowny

- W tym sposobie opisujemy co takiego wykonuje nasz program, tworzymy taką teoretyczną instrukcje jego działania.
  Wyznacz oraz wyświetl największą liczbę z trzech podanych liczb całkowitych, naturalnych

- Po wczytaniu każdej liczby całkowitej naturalnej a, b i c sprawdzaj, czy jest ta liczba większa od liczby zapisanej jako „największa”. Jeżeli dana liczba jest większa od tymczasowej liczby największej to przypisz tą liczbę jako największą. Na koniec działania programu wyświetl największą liczbę.

Lista kroków

W tym sposobie opisujemy kolejno kroki, jakie program ma kolejno wykonywać. Odnosząc się do przykładowej specyfikacji problemu algorytmicznego, lista kroków wygląda następująco:

1. Rozpocznij program
2. Pobierz pierwszą liczbę ***a***
3. Zapamiętaj pobraną liczbę jako ***największa***
4. Pobierz drugą liczbę ***b***
5. Jeżeli pobrana liczba jest większa od liczby największej, zapamiętaj pobraną liczbę jako ***największa***
6. Pobierz trzecią liczbę ***c***
7. Jeżeli pobrana liczba jest większa od liczby największej, zapamiętaj pobraną liczbę jako ***największa***
8. Wyświetl liczbę ***największa***
9. Zakończ program

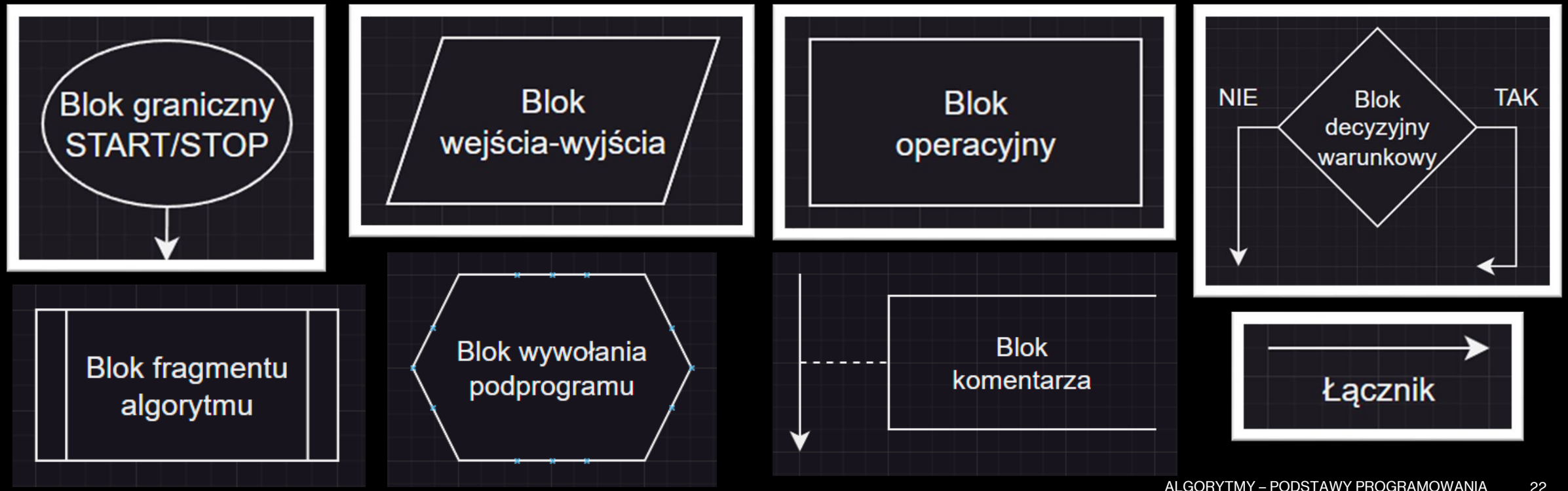
Pseudokod

- Jest to sposób zapisu algorytmu za pomocą kodu, który może zrozumieć każdy programista nieważne jakiego języku. Ma ono być przejrzyste oraz zrozumiałe.

```
START
        L, Max <- 0
        FOR i <- 1, i <= 3 DO
                INPUT L
                IF L > Max THEN
                        Max <- L
                END IF
        END FOR
        PRINT Max
END
```

Schemat blokowy

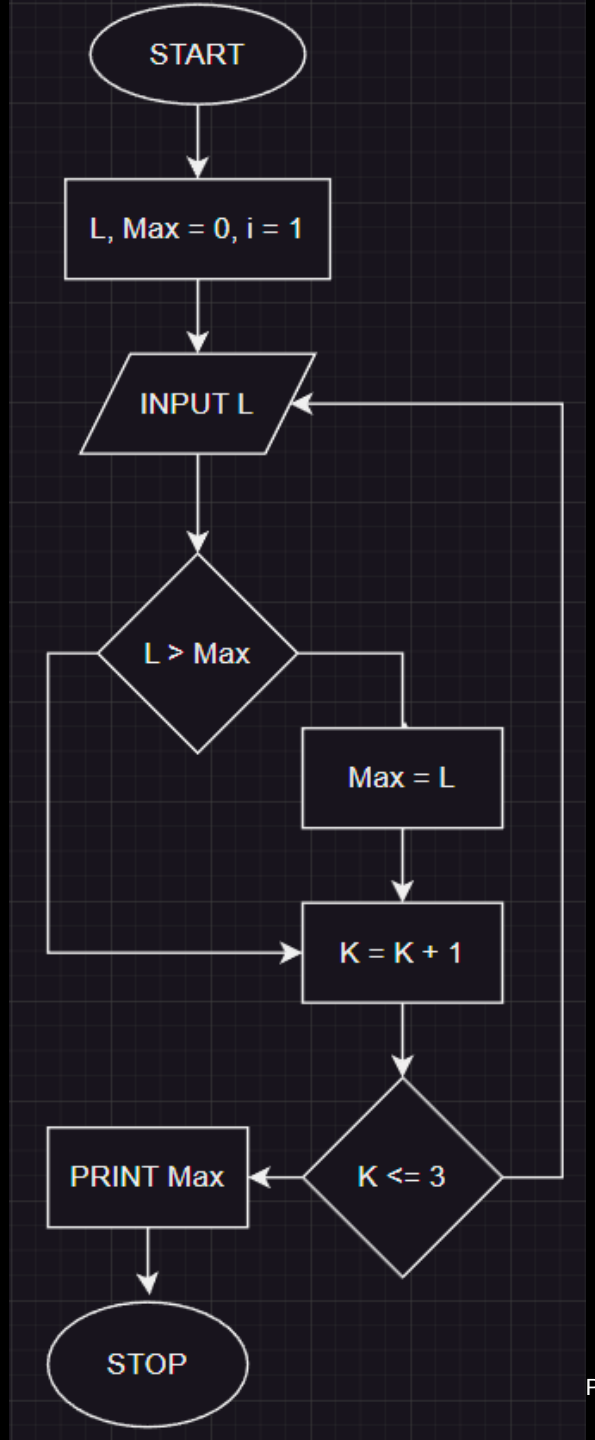
- W tym przypadku opis algorytmu tworzymy za pomocą bloków. Każdy kształt posiada jakąś funkcje, te z ramką na biało są podstawowymi elementami budowy schematu blokowego:



Schemat blokowy

- Schemat blokowy stworzony na podstawie pseudokodu z małymi zmianami:

```
START
        L, Max <- 0
        FOR i <- 1, i <= 3 DO
                INPUT L
                IF L > Max THEN
                        Max <- L
                END IF
        END FOR
        PRINT Max
END
```



Kod problemu algorytmicznego

```
int l, Max = 0;
for(int i = 1; i <= 3; i++){
	l = Console.ReadLine();
	if(l > Max){
		Max = l;
	}
}
Console.WriteLine(Max);
```

Przykładowe dane: 12, 48, 16.
Output: 48

Zadanie utrwalające

- Zapisz listę kroków, schemat blokowy oraz pseudokod dla tego problemu algorytmicznego:
  + Pobierz wartości *a* oraz *b* i oblicz obwód oraz pole prostokąta, a na koniec wyświetl wynik obliczeń dla obwodu oraz pola prostokąta.

  + Do wykonania zadania wykorzystaj: Visual Studio Code, strona Draw.io