

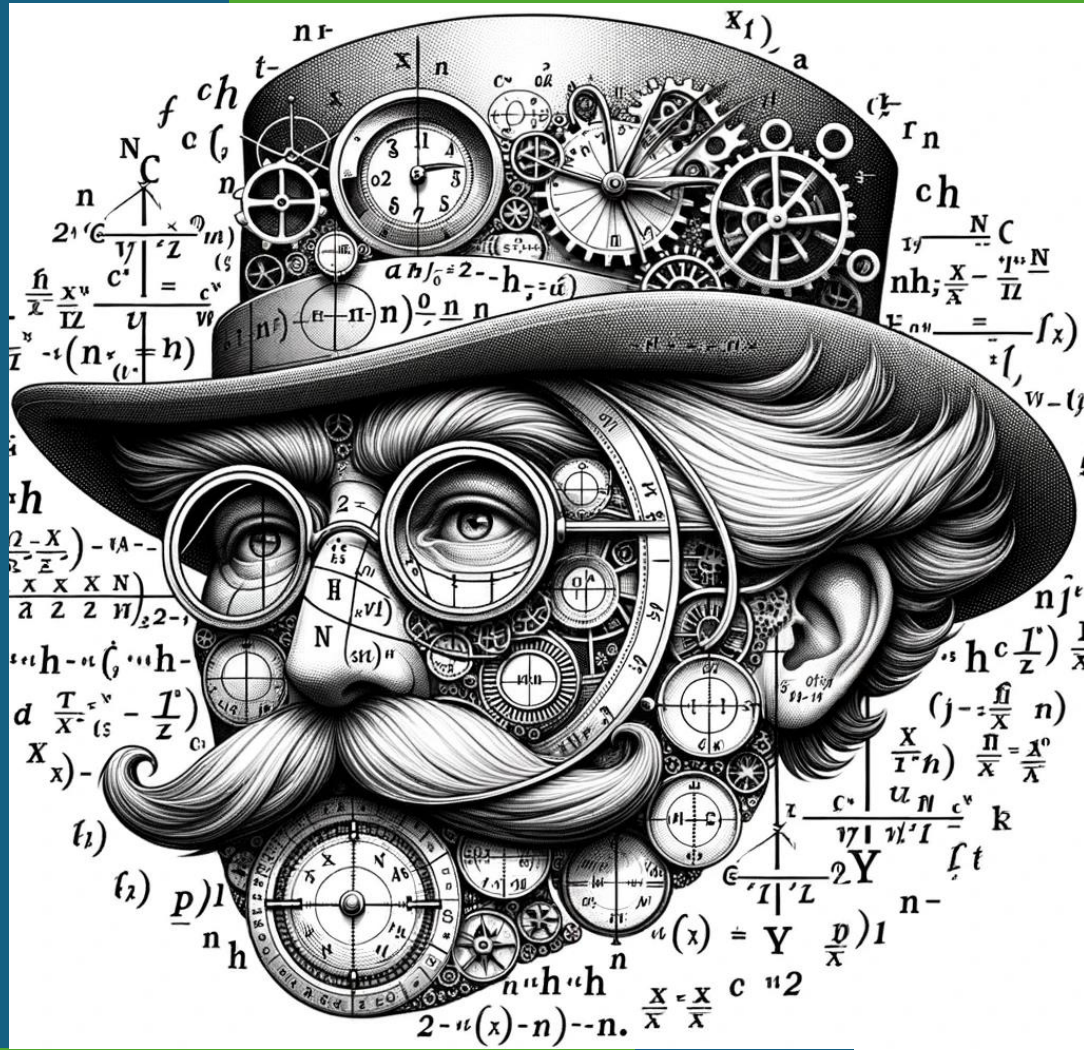
Temat 11.

dr inż. Michał Malinowski

<https://www.drmalinowski.edu.pl>

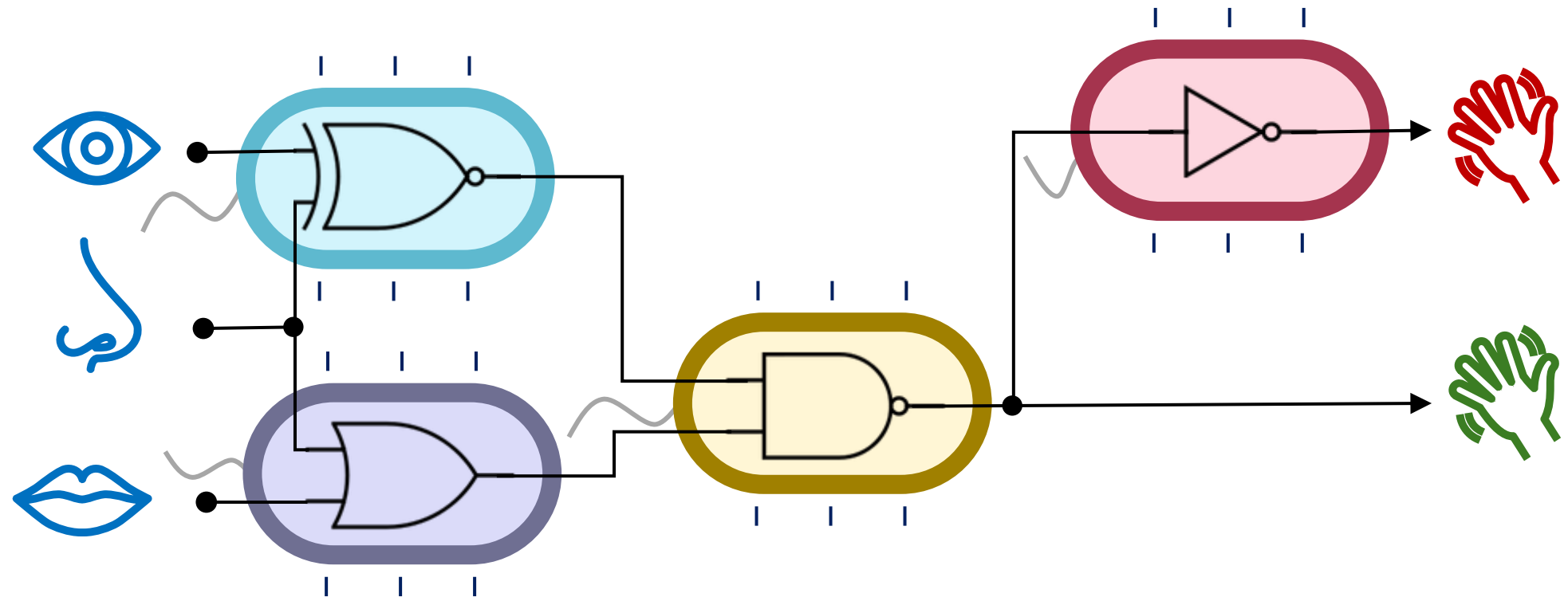
Algorytmy i struktury danych

WPROWADZENIE
DO INFORMATYKI



- Struktury danych
- Operacje na strukturach danych
- Algorytm
- Rodzaje kroków algorytmów
- Złożoność obliczeniowa
- Złożoność czasowa
- Złożoność pamięciowa
- Analiza złożoności algorytmów
- Podział języków programowania
- Bezpieczne języki programowania
- Wyrażenia regularne
- Funkcja haszująca
- Full-Stack

Biologiczne obliczenia logiczne

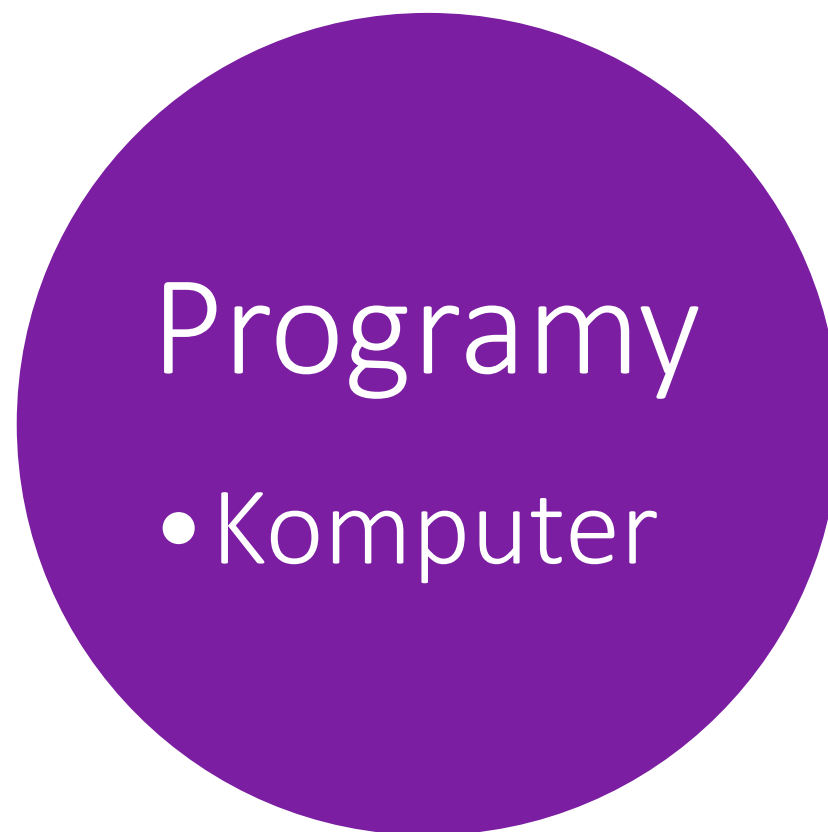
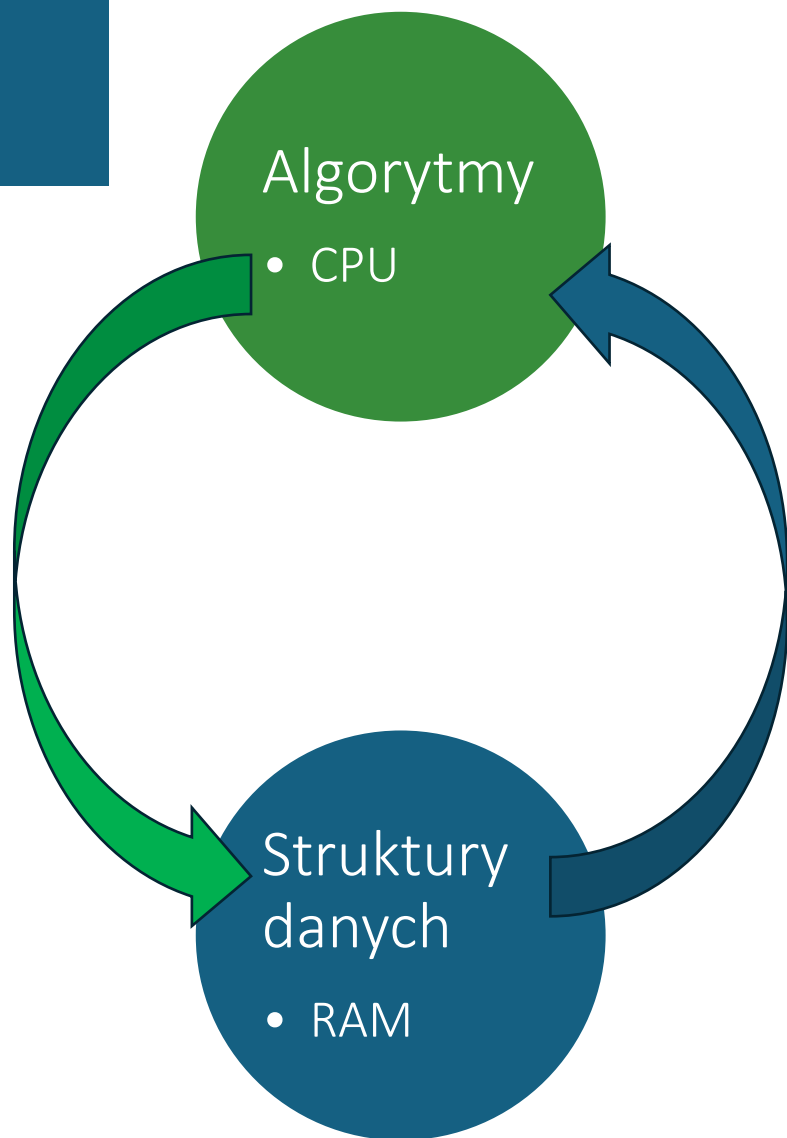




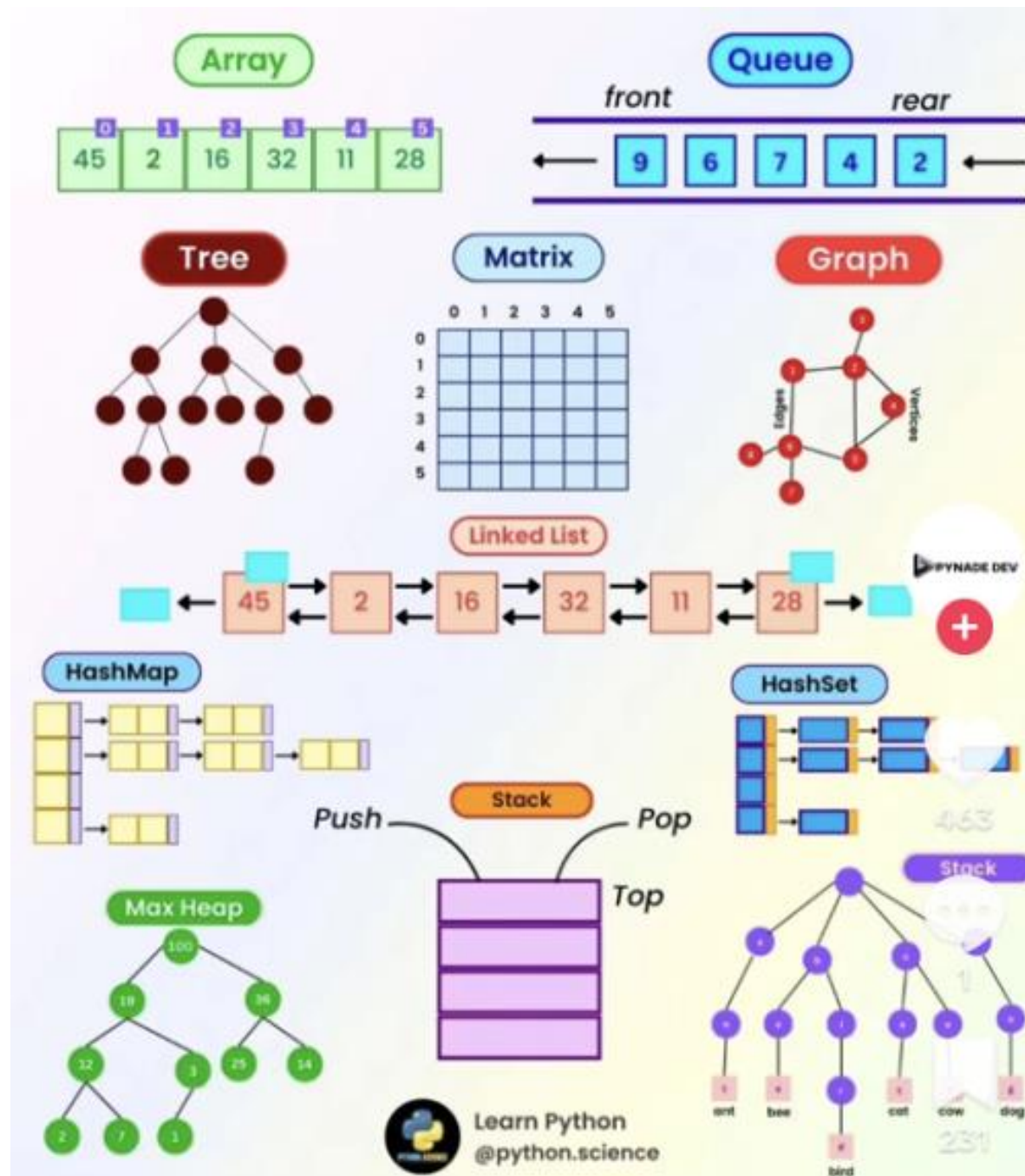
***Musimy myśleć o
programowaniu jak
o fundamentalnej
umiejętności w XXI w.
Porównałbym to do
czytania, pisania, liczenia.***

Września 2023

Miron Mironiuk, założyciel Cosmose.AI,
pomysłodawca programu "Programowanie = nasz drugi język"



Co to jest ?



Struktury danych

Struktura danych

to sposób organizacji i przechowywania danych w komputerze, który umożliwia wydajny dostęp i manipulację tymi danymi.

Co dają?

Optymalizacja

Pozwalają na efektywne wykonywanie operacji na danych, co przekłada się na wydajność programów.

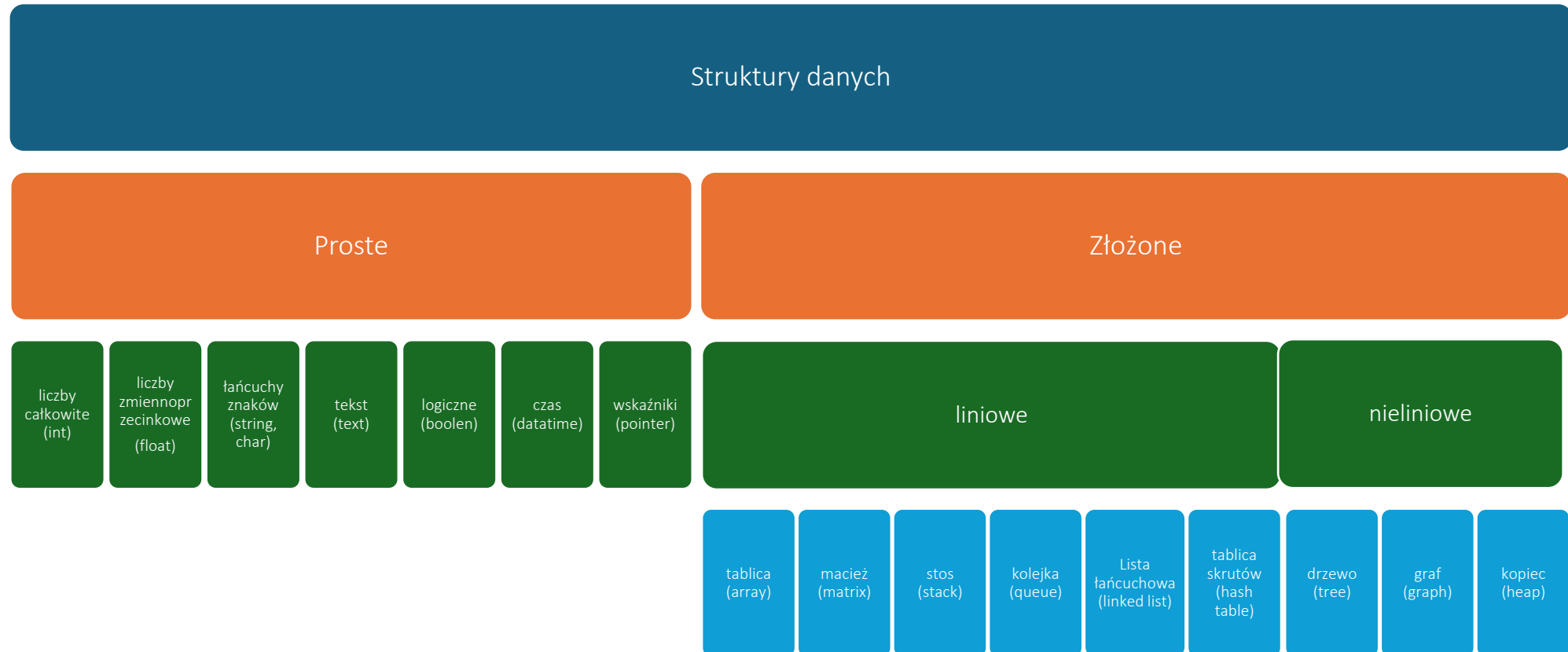
Organizacja

Pomagają w logicznym i efektywnym przechowywaniu danych.

Skalowalność

Umożliwiają dostosowanie się do rosnącej ilości danych.

Struktury danych

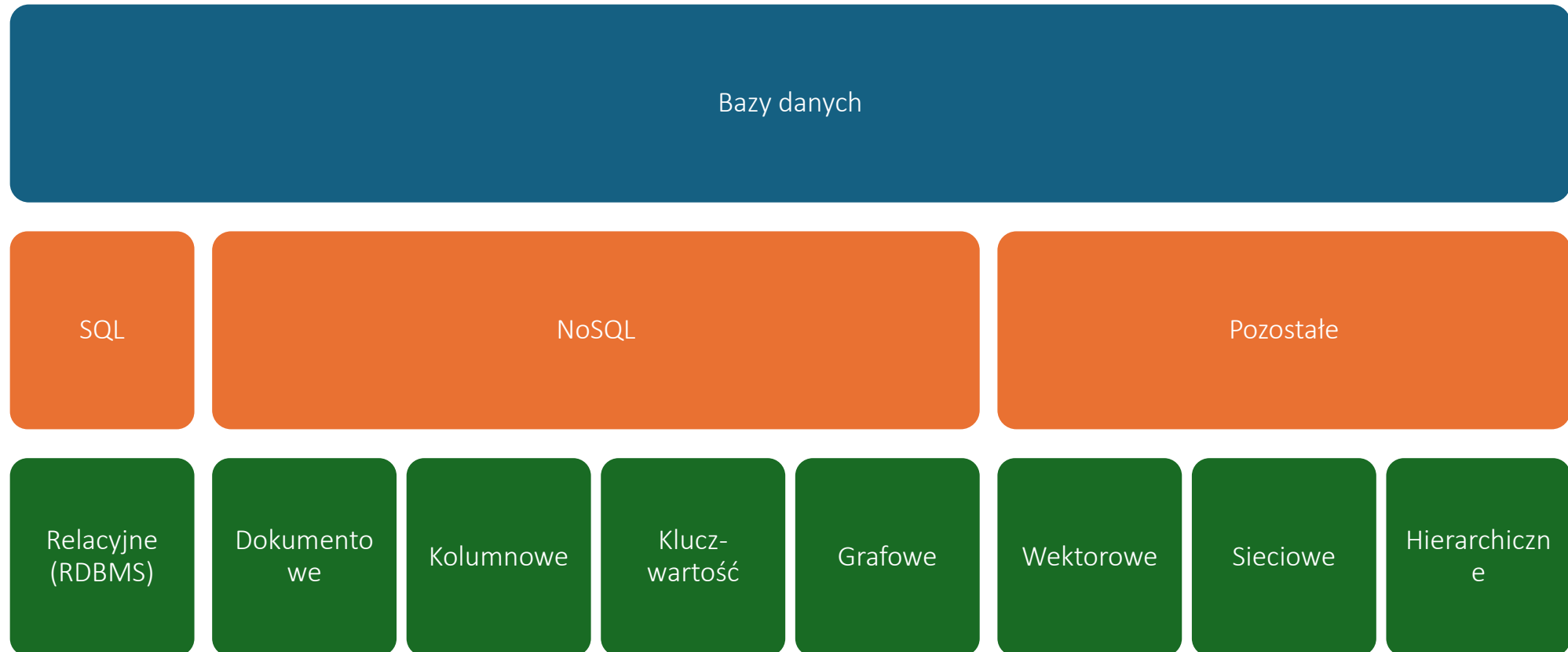


Struktury danych

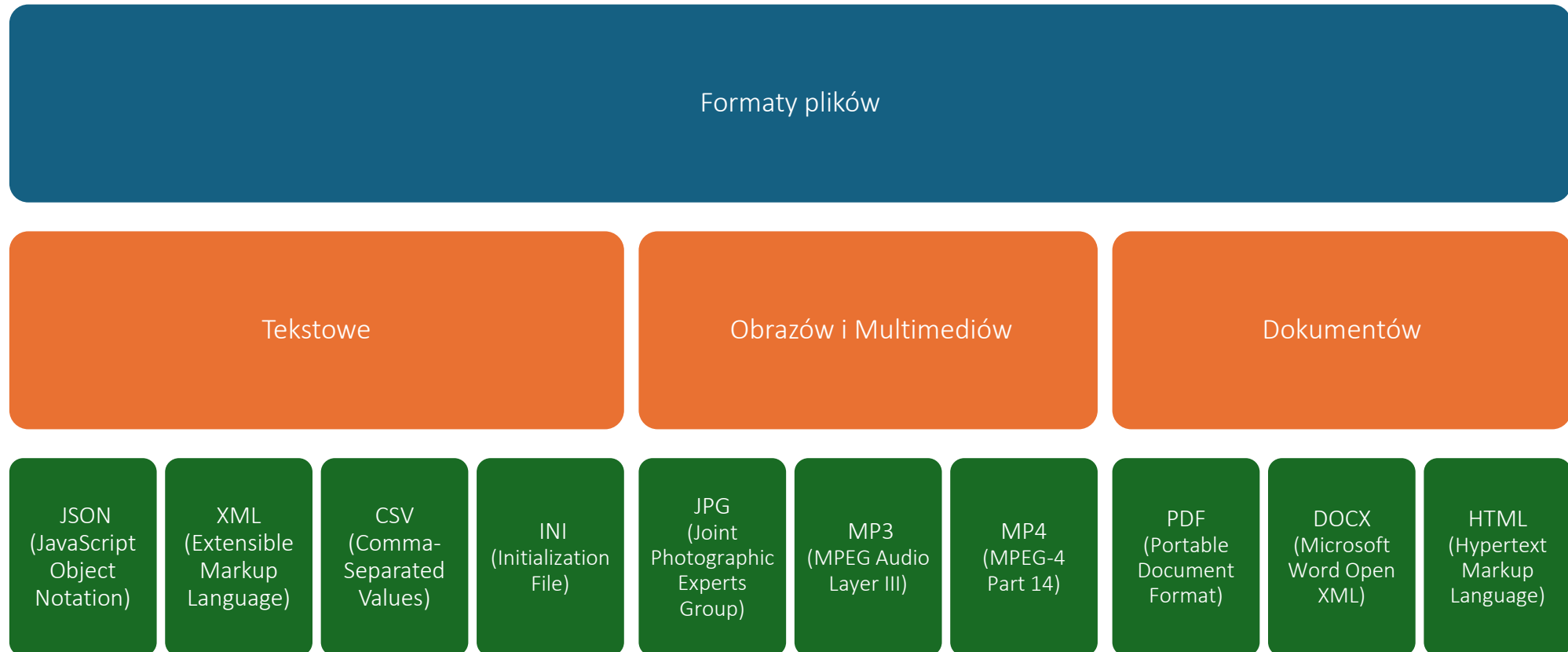
Alokacja pamięci dla struktury

Adresy komórki pamięci	Wartość w komórce	Wartość dla algorytmu	Struktura danych
0xE0D1E			Liczba całkowita (4 bajty)
0xE0D1F	10010010	2345	
0xE0D20	10010001		
0xE0D21	00000000		
0xE0D22	00000000		
0xE0D23	00000011	3	Łańcuch znaków (4 bajty)
0xE0D24	01000001	A	
0xE0D25	01001100	L	
0xE0D26	01000001	A	
0xE0D27			

Struktury danych



Struktury danych



Operacje na strukturach danych

Operacje na strukturach danych

Dodawanie (Insert)

Dodawanie nowych elementów do struktury danych.

Usuwanie (Delete)

Usuwanie istniejących elementów ze struktury danych.

Dostęp (Access)

Pobieranie danych lub odczyt danych z określonego miejsca w strukturze danych.

Operacje na strukturach danych

Operacje na strukturach danych

Modyfikacja (Modification)

Zmiana wartości lub modyfikacja istniejących danych w strukturze danych.

Wyszukiwanie (Search)

Przeszukiwanie struktury danych w celu znalezienia określonego elementu lub wartości.

Sortowanie (Sorting)

Uporządkowanie elementów w strukturze danych według określonego porządku.

Operacje na strukturach danych

Operacje na strukturach danych

Filtrowanie (Filtering)

Wybieranie tylko tych elementów z danej struktury, które spełniają określone kryteria.

Przeszukiwanie (Traversal)

Przechodzenie przez wszystkie elementy w strukturze danych w określonej kolejności.

Łączenie (Joining)

Łączenie dwóch lub więcej struktur danych na podstawie określonych kryteriów.

Operacje na strukturach danych

Operacje na strukturach danych

Dzielenie (Division)

Dzielenie struktury danych na podstruktury lub fragmenty w celu podziału danych na mniejsze części.

Kopiowanie (Copying)

Tworzenie kopii struktury danych lub jej fragmentu w celu dalszego przetwarzania lub analizy.

Optymalizacja (Optimization)

Wykonywanie operacji, które zmniejszają zużycie pamięci lub czasu wykonania operacji na danej strukturze danych.

Algorytm

Algorytm

To skończony zestaw kroków służących do rozwiązania konkretnego problemu.

Założenia dla algorytmu

Poprawność

Algorytm musi rozwiązywać zadanie prawidłowo, dając oczekiwane wyniki.

Skończoność

Algorytm powinien działać w rozsądnym czasie, niezależnie od rozmiaru danych.

Uniwersalność

Algorytm powinien być jasny i zrozumiały dla różnych odbiorców.

Jednoznaczność

Algorytm powinien zwracać oczekiwany wynik.

Algorytmy

Algorytmy

Podstawowe

Grafowe

Dynamiczne

Geometryczne

Szyfrowania

Przetwarzania
Obrazów

Przetwarzania
Języka
Naturalnego

Równoległe i Rozproszone

Sztucznej Inteligencji i Uczenia
Maszynowego

sortowania

wyszukiwania

przeszukiwania
grafów

szukania dróg

programowania
dynamicznego

geometrii
obliczeniowej

grafiki
komputerowej

kryptograficzne

przetwarzania
obrazów
cyfrowych

analizy tekstu

równoległe

rozproszone

uczenia
maszynowego

genetyczne

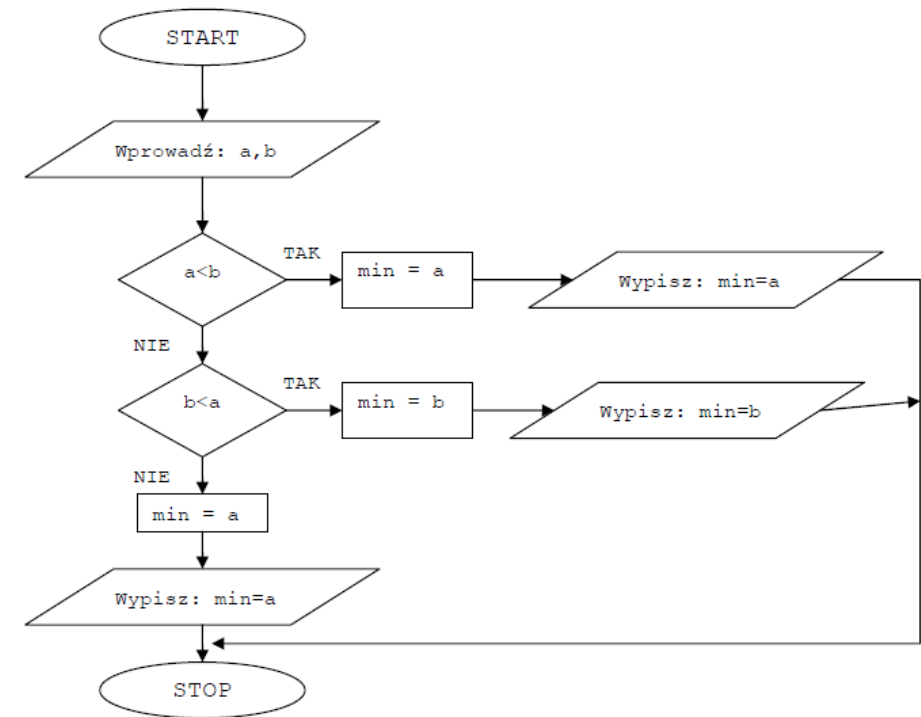
Algorytmy

Krok algorytmu

to podstawowa operacja lub instrukcja, która jest wykonywana w trakcie rozwiązywania problemu lub osiągnięcia określonego celu za pomocą algorytmu.

Metody zapisu kroków algorytmu

- w postaci opisu słownego,
- w postaci listy kroków,
- w postaci schematu blokowego (postać graficzna algorytmu),
- za pomocą jednego z języków formalnych.



Algorytmy

Problem

napisz słownie algorytm obliczania sumy komórek z tablicy o długości n

Opis słowny

Rozpocznij od ustawienia zmiennej o nazwie "suma" na zero. Ustal również zmienną "i" na zero, która będzie służyła jako indeks do iterowania przez tablicę. Następnie, dopóki zmienna "i" jest mniejsza od n , wykonuj poniższe kroki:

1. Dodaj do zmiennej "suma" wartość komórki tablicy o indeksie "i".
2. Zwiększ wartość zmiennej "i" o jeden.

Po przejściu przez wszystkie komórki tablicy, zwróć wartość zmiennej "suma" jako wynik algorytmu. Algorytm kończy swoje działanie po zwróceniu wyniku.

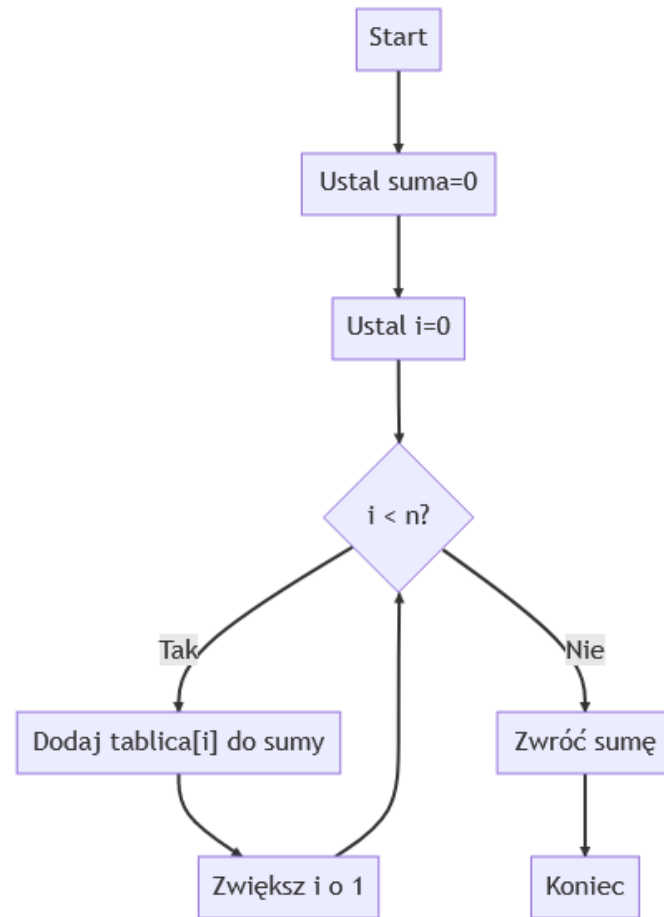
Algorytmy

Lista kroków

1. Ustal zmienną "suma" na 0.
2. Ustal zmienną "i" na 0.
3. Sprawdź, czy $i < n$.
Jeśli tak, przejdź do kroku 4.
Jeśli nie, przejdź do kroku 6.
4. Dodaj wartość komórki tablicy o indeksie i do zmiennej "suma".
5. Zwiększ wartość i o 1 i wróć do kroku 3.
6. Zwróć wartość zmiennej "suma" jako wynik algorytmu.
7. Zakończ algorytm.

Algorytmy

Schemat blokowy



Algorytmy

Język formalny (pseudokod)

suma $\leftarrow$ 0

n $\leftarrow$ Długość(tablica)

dla i od 0 do n-1 wykonaj

 suma $\leftarrow$ suma + tablica[i]

koniec dla

zwróć suma

Rodzaje kroków algorytmów

Rodzaje kroków algorytmów

Wejście (Input):	Pozyskaj dane wejściowe, które są niezbędne do rozwiązania problemu. Mogą to być dane użytkownika, parametry funkcji, pliki, itp.
Wyjście (Output):	Zwróć lub wyświetl wynik działania algorytmu. Wynik może być liczbą, tekstem, grafiką lub innym rodzajem danych.
Deklaracje (Declarations):	Deklaracje to tworzenie zmiennych lub struktur danych, które są używane w trakcie przetwarzania. W językach programowania deklaracje zmiennych są często potrzebne.

Rodzaje kroków algorytmów

Rodzaje kroków algorytmów

Przetwarzanie (Processing):

Wykonaj operacje i obliczenia na danych wejściowych, które prowadzą do rozwiązania problemu.

Operacje arytmetyczne

Operacje logiczne

Operacje na strukturach danych

Warunki (Conditions):

Algorytmy często zawierają warunki, które decydują o tym, jakie kroki należy wykonać w zależności od określonych sytuacji. To może obejmować instrukcje warunkowe "if", "else", "switch" itp.

Pętle (Loops):

Jeśli algorytm wymaga wielokrotnego powtarzania określonych kroków, stosuje się pętle. Przykłady to pętla "for", "while", "do...while".

Funkcje (Functions):

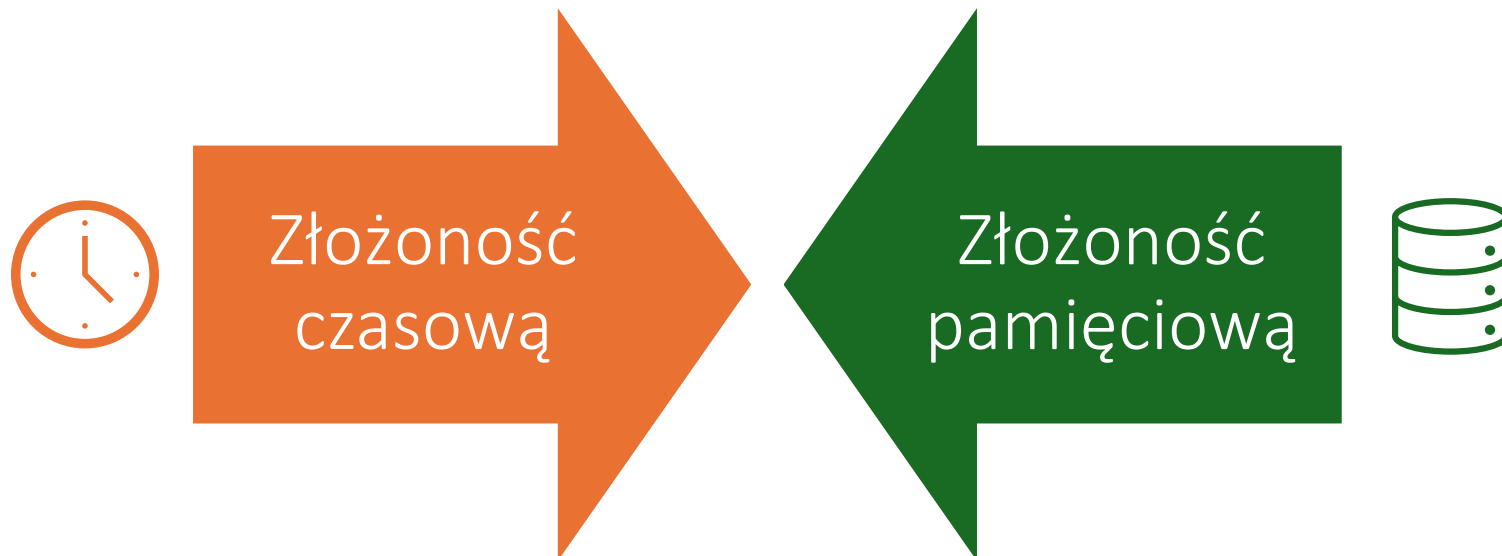
Jeśli algorytm jest bardziej złożony, może zawierać funkcje lub procedury, które grupują powtarzalne kroki w jednostki, co ułatwia zarządzanie i zrozumienie kodu.

Złożoność obliczeniowa

Złożoność obliczeniowa

jest to miara określająca zapotrzebowania danego rozwiązania na zasoby systemu w którym ma być rozwiązany

Zasobami branymi pod uwagę są czas i pamięć. Rozróżnia się więc dwa rodzaje złożoności algorytmów:



Złożoność czasowa

Złożoność czasowa

jest to funkcję opisującą ilość operacji (lub kroków) wykonanych przez algorytm w zależności od wielkości danych wejściowych.

Liczba sekund, godzin, dni .. najczęściej liczba operacji jakie należy wykonać, aby problem został rozwiązany.

Pomaga ocenić, ile czasu zajmuje wykonanie algorytmu w zależności od rozmiaru danych.

Pozwala przewidzieć, jak algorytm zachowuje się w miarę wzrostu ilości danych wejściowych.

Złożoność pamięciowa

Złożoność pamięciowa

odnosi się do ilości pamięci (przestrzeni) wymaganej przez algorytm w zależności od wielkości danych wejściowych.

Liczba bajtów, kilobajtów, megabajtów pamięci RAM i/lub pamięci dyskowej. Również liczba plików, łącz i innych zasobów pamięciowych jakie są niezbędne do realizacji algorytmu.

Pomaga ocenić, ile
pamięci zajmuje
algorytm.

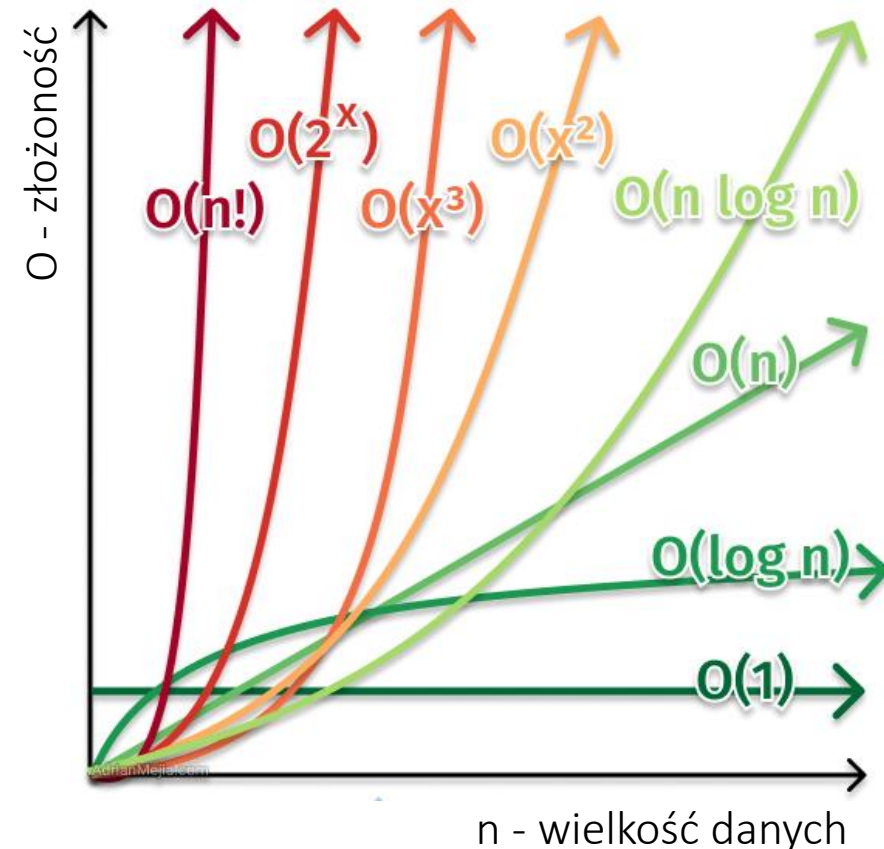
Ważna w sytuacjach,
gdzie ograniczenia
dotyczą dostępnej
przestrzeni pamięci.

Analiza złożoność algorytmów

Notacja O (Wielkie O, Big O)

Notacja O jest matematycznym sposobem opisanie górnej granicy złożoności algorytmu w najgorszym przypadku.

Pozwala klasyfikować algorytmy na podstawie ich zachowania względem rozmiaru danych i dzielić na klasy złożoności obliczeniowej.



Analiza złożoność algorytmów

$O(1)$ - Stała złożoność czasowa:

Oznacza, że czas wykonania algorytmu jest stały i nie zależy od rozmiaru danych wejściowych. Jest to idealna sytuacja, ale rzadko występuje w praktyce.

$O(n)$ - Liniowa złożoność czasowa:

Oznacza, że czas wykonania algorytmu rośnie liniowo wraz z rozmiarem danych wejściowych.
Przykład: Przeszukiwanie liniowe listy.

$O(\log n)$ - Logarytmiczna złożoność czasowa:

Oznacza, że czas wykonania algorytmu rośnie logarytmicznie wraz z rozmiarem danych wejściowych.
Przykład: Wyszukiwanie binarne w posortowanej liście.

$O(n \log n)$ - Złożoność czasowa $n \log n$:

Jest charakterystyczna dla wielu efektywnych algorytmów sortowania.
Przykład: Algorytm QuickSort.

Analiza złożoność algorytmów

$O(n^2)$ - Kwadratowa złożoność czasowa:

Oznacza, że czas wykonania algorytmu rośnie kwadratowo wraz z rozmiarem danych wejściowych.

Przykład: Sortowanie bąbelkowe.

$O(2^n)$ - Wykładnicza złożoność czasowa:

Oznacza, że czas wykonania algorytmu rośnie wykładniczo wraz z rozmiarem danych wejściowych.

Jest to złożoność, której należy unikać w praktyce ze względu na jej ekstremalną rosnącą złożoność.

Przykład: Obliczenie ciągu Fibonacciego

$O(n!)$ - Silnia złożoność czasowa:

Oznacza, że czas wykonania algorytmu rośnie zgodnie z funkcją silnia wraz z rozmiarem danych wejściowych. Jest to najgorszy przypadek i najbardziej nieefektywna złożoność.

Przykład: Problem komiwojażera

Analiza złożoność algorytmów

Porównanie czasów wykonania algorytmów różnych typów przy założeniu, że dla każdy z nich $n = 1$ wykonuje się $10^{-6}s$

n	10	20	30	40	50	60
$O(n)$	0,00001s	0,00002s	0,00003s	0,00004s	0,00005s	0,00006s
$O(n^2)$	0,0001s	0,0004s	0,0009s	0,0016s	0,0025s	0,0036
$O(n^3)$	0,001s	0,008s	0,027s	0,064s	0,125s	0,216s
$O(2^n)$	0,001s	1,048s	17,9min	12,7dni	35,7lat	366w
$O(3^n)$	0,059s	58min	6,5lat	3855w	$227 \cdot 10^6 w$	$1,3 \cdot 10^{13} w$

Podział języków programowania

Kryterium	Kategoria	Przykłady języków
Poziom abstrakcji	Języki niskiego poziomu	Asembler, język maszynowy
	Języki wysokiego poziomu	C, Java, Python
Paradygmaty programowania	Proceduralne	C, Fortran
	Obiektowe	Java, C++, Python
	Funkcyjne	Haskell, Lisp
	Logiki	Prolog
	Skryptowe	JavaScript, Python, Ruby
Sposób wykonywania	Kompilowane	C, C++, Rust
	Interpretowane	Python, Ruby, JavaScript
	JIT (Just-In-Time)	Java (JVM), C# (CLR)
Zastosowanie	Ogólnego przeznaczenia	Python, Java, C++
	Specjalistyczne	MATLAB, SQL, R
Składnia	C-like	C++, Java, C#
	Pascal-like	Delphi, Ada
	ML-like	Haskell, F#
Inne kryteria	Deklaratywne	SQL, HTML
	Imperatywne	C, Java
	Statycznie typowane	Java, C++
	Dynamicznie typowane	Python, JavaScript

Biały Dom pilnie domaga się, aby programiści używali języków programowania bezpiecznych dla pamięci.

Luty 2024

Biuro Krajowego Dyrektora ds. Cyberbezpieczeństwa (ONCD)
w Białym Domu



Bezpieczne języki programowania

Automatyczne zarządzanie pamięcią

Eliminuje błędy związane z ręcznym zarządzaniem pamięcią, takie jak wycieki pamięci i podwójne zwalnianie pamięci, co zapobiega wielu potencjalnym problemom bezpieczeństwa.

Bezpieczne typowanie

Pomaga wykrywać błędy na wczesnym etapie (podczas kompilacji), zapobiegając nieprawidłowym operacjom na danych, które mogłyby prowadzić do błędów

Sprawdzanie granic

Dla tablic i innych struktur danych zapobiega przepełnieniom bufora, które mogą prowadzić do nadpisania danych i potencjalnych luk w bezpieczeństwie.

Zapobieganie wstrzyknięciu kodu

Uniemożliwia wstrzyknięcie i wykonanie nieautoryzowanego kodu, na przykład przez odpowiednie sanitowanie danych wejściowych i stosowanie bezpiecznych API.

Model własności i zarządzanie dostępem

Stosowanie modeli własności i pożyczania, które na poziomie kompilacji zapewniają, że dostęp do danych jest zarządzany w sposób bezpieczny, eliminując wyścigi danych i inne błędy związane z konkurencyjnym dostępem.

Immutability (niemutowalność)*

Wymuszanie stosowania niemutowalnych struktur danych pomaga w zapobieganiu błędom wynikającym ze zmian stanu, co jest szczególnie ważne w aplikacjach współbieżnych i rozproszonych.

*Immutability, czyli niemutowalność, to właściwość obiektów (zmiennych) w programowaniu, która nie pozwala na zmianę ich stanu po utworzeniu

Bezpieczne języki programowania

Bezpieczne języki	Pośrednie języki	Niebezpieczne języki
Rust	JavaScript	C
Swift	TypeScript	C++
Go	Kotlin	Assembly
Java	Scala	
C#	Python	
Haskell	Ruby	
Elixir	PHP	

Znaki i wartości kodu ASCII

Standardowy Kod do Przetwarzania Informacji (ASCII) to zestaw znaków i ich odpowiadających liczbowych wartości.

- Obejmuje 128 znaków, reprezentowanych przez liczby od 0 do 127.
- Zawiera znaki kontrolne, cyfry, litery i znaki specjalne.

Znak	Wartość
['\n', '\t', ' ', '!', '"', '#', '\$', '%', '&', "'", '(', ')']	['10', '9', '32', '33', '34', '35', '36', '37', '38', '39', '40', '41']
['*', '+', ',', '-', '.', '/', '0', '1', '2', '3', '4', '5']	['42', '43', '44', '45', '46', '47', '48', '49', '50', '51', '52', '53']
['6', '7', '8', '9', ':', ';', '<', '=', '>', '?', '@', 'A']	['54', '55', '56', '57', '58', '59', '60', '61', '62', '63', '64', '65']
['B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', 'K', 'L', 'M']	['66', '67', '68', '69', '70', '71', '72', '73', '74', '75', '76', '77']
['N', 'O', 'P', 'Q', 'R', 'S', 'T', 'U', 'V', 'W', 'X', 'Y']	['78', '79', '80', '81', '82', '83', '84', '85', '86', '87', '88', '89']
['Z', '[', '\', ']', '^', '_', '`', 'a', 'b', 'c', 'd', 'e']	['90', '91', '92', '93', '94', '95', '96', '97', '98', '99', '100', '101']
['f', 'g', 'h', 'i', 'j', 'k', 'l', 'm', 'n', 'o', 'p', 'q']	['102', '103', '104', '105', '106', '107', '108', '109', '110', '111', '112', '113']
['r', 's', 't', 'u', 'v', 'w', 'x', 'y', 'z', '{', ' ', '}']	['114', '115', '116', '117', '118', '119', '120', '121', '122', '123', '124', '125']
['~']	['126']

Wyrażenia regularne

Wyrażenia regularne

są wzorcami służącymi do dopasowywania kombinacji znaków w ciągach tekstowych. Wykorzystywane są one do wyszukiwania, walidacji, i manipulacji tekstem.

```
let wzorzec = /ab+c/i;  
console.log('Czy wzorzec pasuje do "Abc"?', wzorzec.test('Abc')); // Wynik: true  
  
let wynik = wzorzec.exec('Pamiętaj o abcd i ABC');  
console.log('Wynik dopasowania:', wynik[0]); // Wynik: "Abc"  
  
console.log('Flagi wzorca:', wzorzec.flags); // Wynik: "i"  
console.log('Tekst wzorca:', wzorzec.source); // Wynik: "ab+c"  
  
let regexGlobalny = /abc/g;  
console.log('Czy wzorzec globalny?', regexGlobalny.global); // Wynik: true
```

Wyrażenia regularne

Symbol	Opis
.	Dopasowuje dowolny pojedynczy znak, z wyjątkiem znaku nowej linii.
^	Dopasowuje początek ciągu znaków lub linii.
\$	Dopasowuje koniec ciągu znaków lub linii.
*	Dopasowuje zero lub więcej wystąpień poprzedzającego elementu.
+	Dopasowuje jedno lub więcej wystąpień poprzedzającego elementu.
?	Dopasowuje zero lub jedno wystąpienie poprzedzającego elementu.
{n}	Dopasowuje dokładnie n wystąpień poprzedzającego elementu.
{n,}	Dopasowuje n lub więcej wystąpień poprzedzającego elementu.
{n,m}	Dopasowuje od n do m wystąpień poprzedzającego elementu.
[abc]	Dopasowuje dowolny pojedynczy znak z podanych (w tym przypadku a, b lub c).
[^abc]	Dopasowuje dowolny pojedynczy znak, który nie jest wymieniony (w tym przypadku nie a, b ani c).
(abc)	Dopasowuje dokładnie sekwencję abc (grupowanie).
\s	Dopasowuje dowolny biały znak, w tym spację, tabulację, znak nowej linii.
\S	Dopasowuje dowolny znak inny niż biały znak.
\d	Dopasowuje dowolną cyfrę (0-9).
\D	Dopasowuje dowolny znak, który nie jest cyfrą.
\w	Dopasowuje dowolny znak alfanumeryczny, w tym cyfrę lub podkreślenie (_).
\W	Dopasowuje dowolny znak, który nie jest znakiem alfanumerycznym ani podkreśleniem.
\b	Dopasowuje granicę słowa, czyli pozycję pomiędzy znakiem \w a \W (lub na początku/koniec ciągu).
\B	Dopasowuje pozycję, która nie jest granicą słowa.
\\	Używany do escapowania znaków specjalnych, np. \\ dopasowuje literał kropki.

```
const emailRegex = /^[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}$/;  
const testEmail1 = "przykładowy.email@example.com";  
const testEmail2 = "nieprawidłowyemail@przykład";  
  
console.log(emailRegex.test(testEmail1)); // Wynik: true  
console.log(emailRegex.test(testEmail2)); // Wynik: false
```

```
const kodPocztowyRegex = /^[0-9]{2}-[0-9]{3}$/;  
const testKodPocztowy1 = "00-123";  
const testKodPocztowy2 = "00123";  
  
console.log(kodPocztowyRegex.test(testKodPocztowy1)); // Wynik: true  
console.log(kodPocztowyRegex.test(testKodPocztowy2)); // Wynik: false
```

```
const peselRegex = /^[0-9]{11}$/;  
const testPesel1 = "85050515836";  
const testPesel2 = "1234567890A";  
  
console.log(peselRegex.test(testPesel1)); // Wynik: true  
console.log(peselRegex.test(testPesel2)); // Wynik: false
```

Funkcja haszująca

Funkcja haszująca (mieszająca)

to funkcja, która przyjmuje ciąg znaków i zwraca unikatowy (zazwyczaj) ciąg znaków o ustalonej długości.

Znane algorytmy haszujące: MD5, SHA1, SHA256

Dla MD5:

- 32 znaki wynikowe HEX (0,1,2...,E,F)
- Liczba kombinacji = $1208925819614629174706176 = 1.21 \times 10^{24}$

wartość	hasz
a	0cc175b9c0f1b6a831c399e269772661
A	7fc56270e7a70fa81a5935b72eacbe29
Litwo ojczyzno moja ty jesteś jak zdrowie	0ab62d3f1aa6412aaaf26fbf43245d0d

Full-Stack

Full-Stack (Developer)

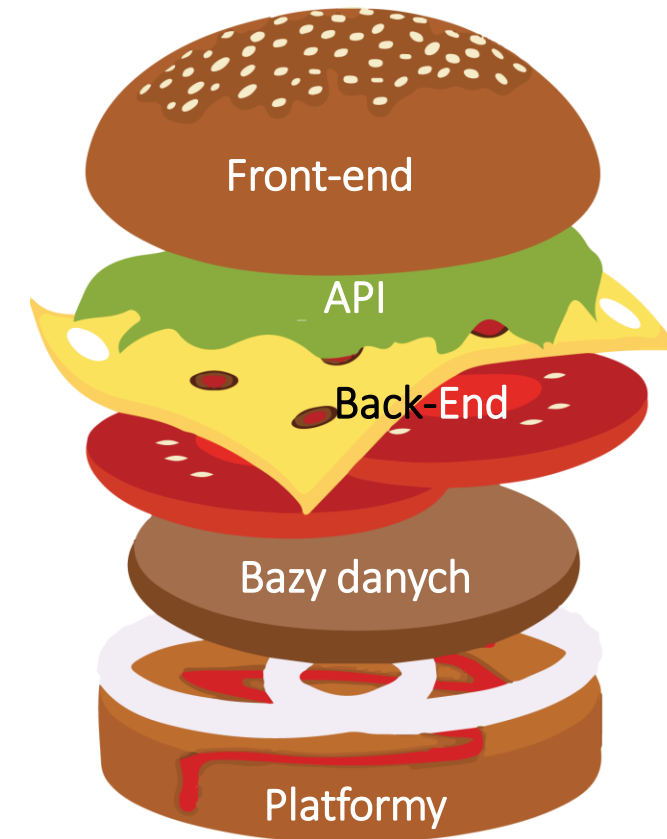
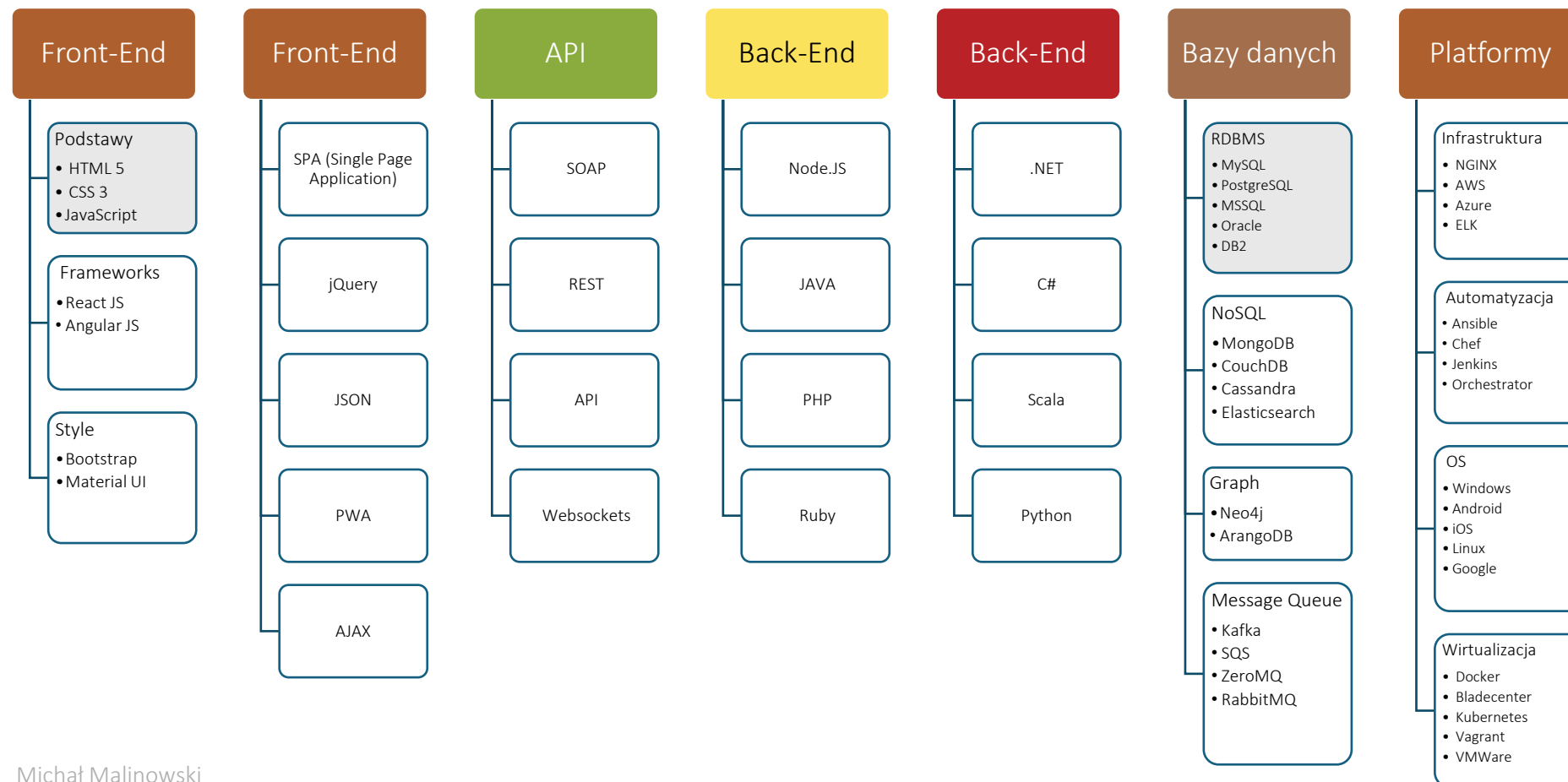
odnosi się do kompletu technologii i umiejętności niezbędnych do projektowania, implementacji oraz zarządzania całą strukturą aplikacji internetowej lub mobilnej.

Obejmuje to zarówno

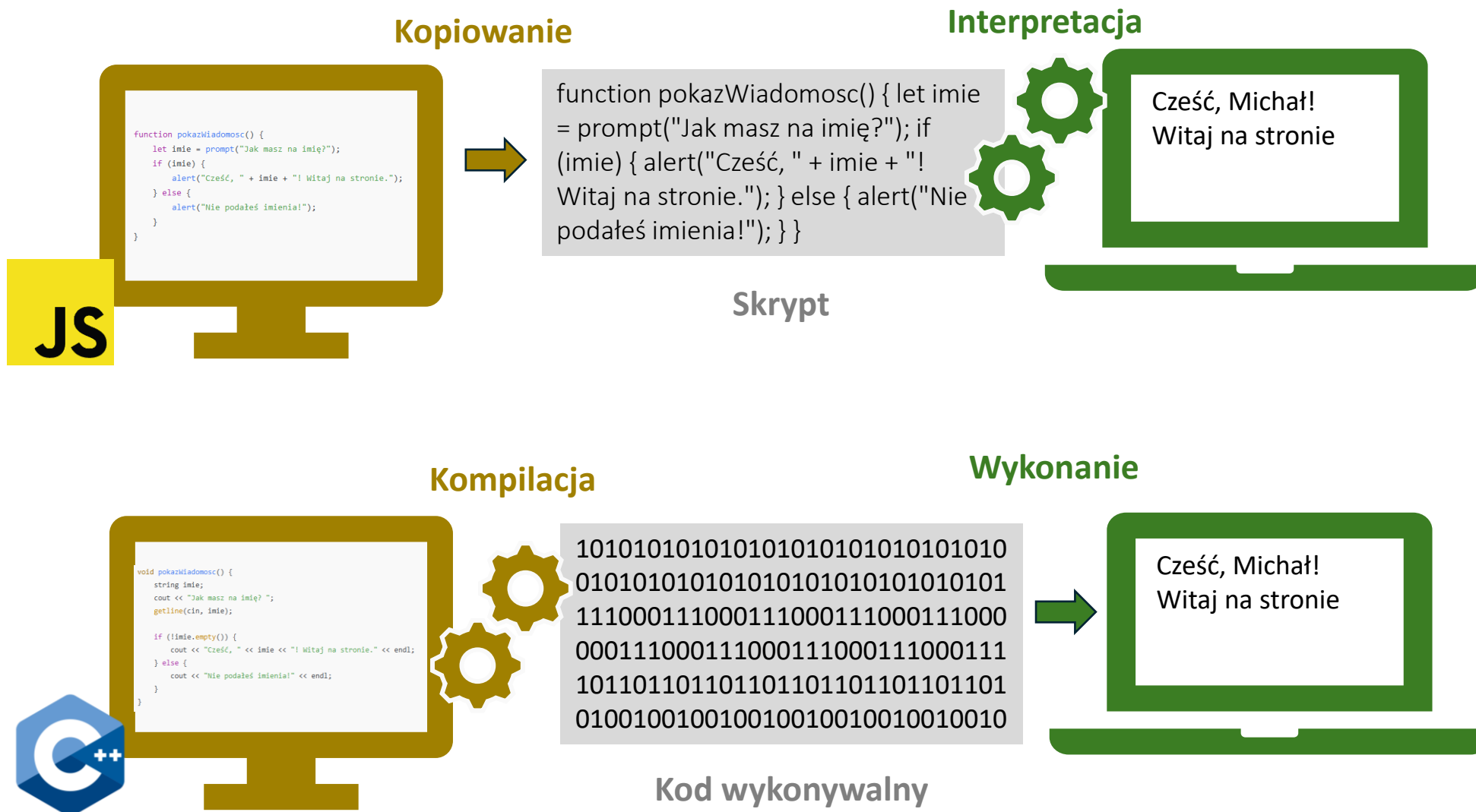
- Front-End, czyli część aplikacji widoczną i dostępną dla użytkownika (interfejs użytkownika),
- Back-End, czyli część serwerową odpowiedzialną za logikę działania aplikacji, przetwarzanie danych i komunikację z bazą danych.

Programista Full-Stack jest więc w stanie samodzielnie stworzyć pełną aplikację webową lub mobilną, od zera do wersji produkcyjnej.

Full-Stack



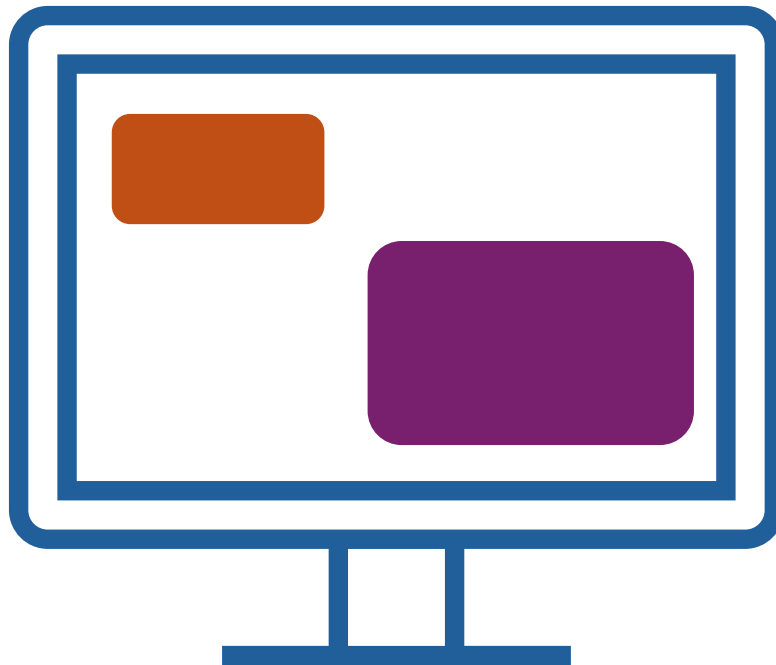
Kompilacja vs Interpretacja kodu programów



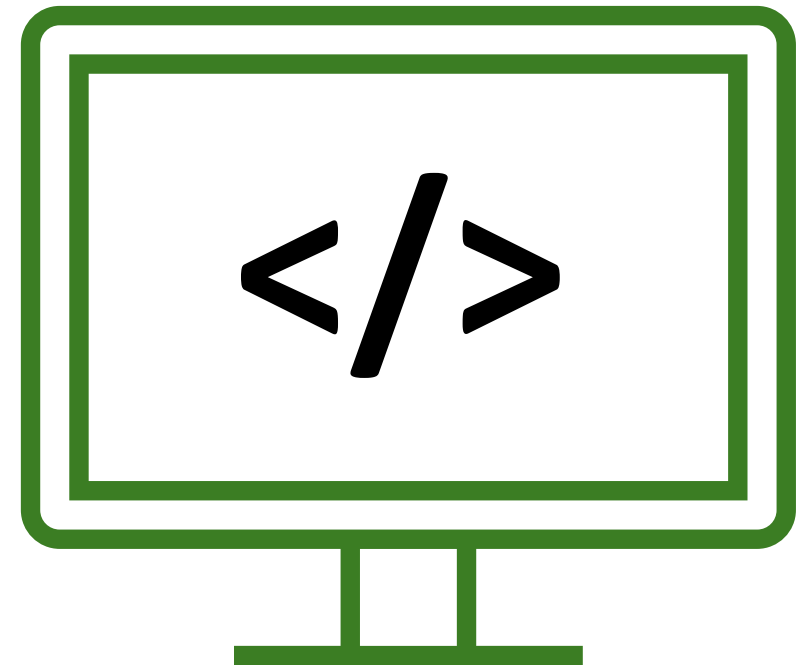
No-code vs Low-code



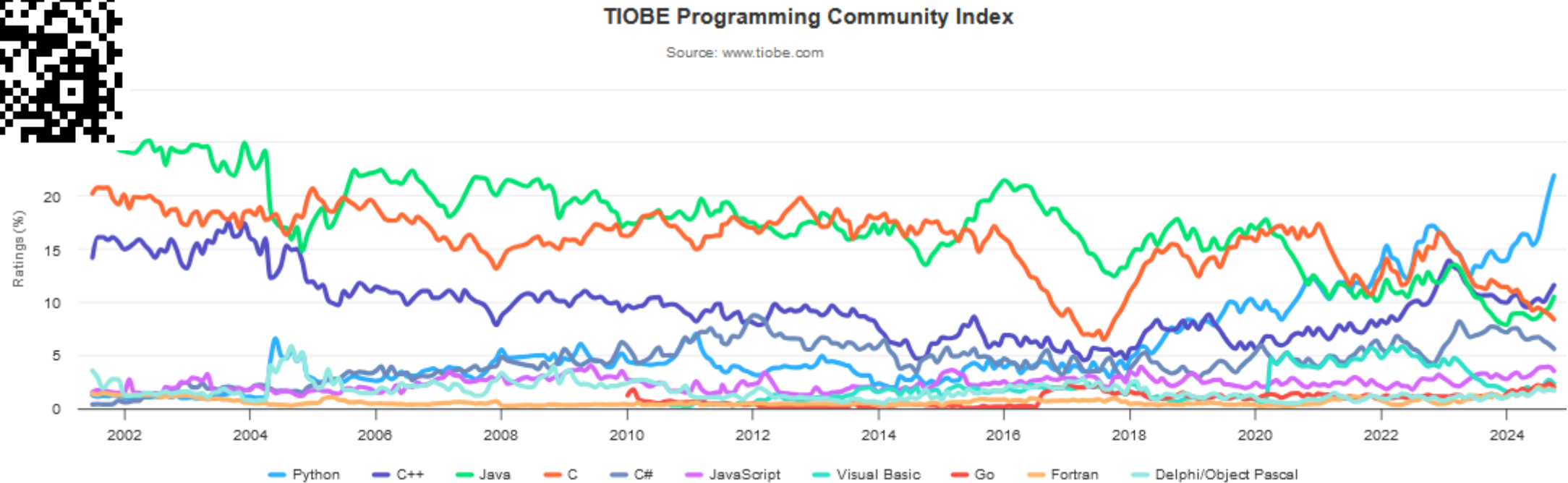
No-code



Low-code



Ranking - Języki programowania



TIOBE Index

<https://www.tiobe.com/tiobe-index/>

Ranking - Języki programowania



Oct 2024	Oct 2023	Change	Programming Language		Ratings	Change
1	1		 Python		21.90%	+7.08%
2	3	^	 C++		11.60%	+0.93%
3	4	^	 Java		10.51%	+1.59%
4	2	v	 C		8.38%	-3.70%
5	5		 C#		5.62%	-2.09%
6	6		 JavaScript		3.54%	+0.64%
7	7		 Visual Basic		2.35%	+0.22%
8	11	^	 Go		2.02%	+0.65%
9	16	^^	 Fortran		1.80%	+0.78%
10	13	^	 Delphi/Object Pascal		1.68%	+0.38%
11	9	v	 SQL		1.64%	-0.15%
12	14	^	 MATLAB		1.48%	+0.22%
13	20	^^	 Rust		1.45%	+0.53%
14	12	v	 Scratch		1.41%	+0.05%
15	8	vv	 PHP		1.21%	-0.69%
16	10	vv	 Assembly language		1.13%	-0.51%
17	17		 R		1.09%	+0.12%
18	19	^	 Ruby		0.99%	+0.07%
19	24	^^	 COBOL		0.99%	+0.23%
20	15	vv	 Swift		0.98%	-0.09%

TIOBE Index

<https://www.tiobe.com/tiobe-index/>

