Zagadnienie do egzaminu inżynierskiego 24/25

Semestr III/Semestr IV

Matematyka 2

1. Rachunek różniczkowy i całkowy funkcji jednej zmiennej rzeczywistej

Rachunek różniczkowy

Rachunek różniczkowy jest jednym z dwóch głównych działów analizy matematycznej (drugim jest rachunek całkowy). Zajmuje się badaniem, jak funkcje zmieniają się w zależności od swoich argumentów. Podstawowe pojęcia rachunku różniczkowego to pochodna, różniczka i gradient.

Definicja pochodnej

Pochodną funkcji f(x) w punkcie x_0 definiujemy jako granicę:

$$f'(x_0) = \lim_{h \to 0} \frac{f(x_0 + h) - f(x_0)}{h}$$

Jest to współczynnik kierunkowy stycznej do wykresu funkcji w punkcie x_0.

Interpretacja:

- Pochodna mierzy prędkość zmian funkcji.
- Jeżeli f'(x)>0, funkcja rośnie w punkcie x.
- Jeżeli f'(x)<0, funkcja maleje w punkcie x.

Różniczka

Różniczka jest liniowym przybliżeniem zmiany funkcji w małym sąsiedztwie punktu. Jej wartość jest dana wzorem:

df=f'(x) dx

gdzie dx to dowolna zmiana argumentu.

3. Reguły różniczkowania

Podstawowe reguły:

1. **Stała:** $\frac{d}{dx}[C] = 0$
2. **Funkcja liniowa:** $\frac{d}{dx}[ax + b] = 0$

3. **Suma/dodawanie:** $\frac{d}{dx}[f(x)+g(x)] = f'(x)+g'(x)$
4. **Iloczyn:** $\frac{d}{dx}[f(x)*g(x)] = f'(x)*g(x)+f(x)g'(x)$
5. **Iloraz:** $\frac{d}{dx}\left[\frac{f(x)}{\mathrm{g(x)}}\right] = \frac{f'(x)*g(x)-f(x)*g'(x)}{q(x)^2}$
6. **Funkcja złożona (reguła łańcuchowa):** $\frac{d}{dx}[f(g(x))] = f'\big(g(x)\big)*g'(x)$

Pochodne wybranych funkcji:

- $\frac{d}{dx}[x^n)] = nx^{n-1}$
- $\frac{d}{dx}[\sin(x)] = \cos(x)$
- $\frac{d}{dx}[\cos(x)] = -sin(x)$
- $\frac{d}{dx}[\ln(x)] = 1/x$
- $\frac{d}{dx}[e^x] = e^x$

Własności i zastosowania pochodnej

- **Monotoniczność funkcji**:
  Funkcja jest rosnąca na przedziale, gdy f'(x)>0, oraz malejąca, gdy f'(x)<0.
- **Ekstrema lokalne**:
  Ekstremum występuje, gdy f'(x) = 0, a druga pochodna f''(x) określa charakter (minimum lub maksimum).
- **Ruch i zmiany**:
  Pochodna opisuje prędkość w ruchu prostoliniowym (pierwsza pochodna położenia względem czasu) oraz przyspieszenie (druga pochodna)

Przykład:
Funkcja $f(x)=x^3-3x^2+2x$

- Pochodna: $f'(x)=3x^2-6x+2$
- Ekstrema: Rozwiązujemy f'(x)=0, co daje x=1 i x=2/3

Rachunek całkowy

Rachunek całkowy zajmuje się obliczaniem całek, które mierzą sumaryczny efekt zmian funkcji, takie jak pole pod wykresem lub całkowity przepływ.

Rodzaje całek

- **Całka nieoznaczona**: Funkcja pierwotna, która jest odwrotnością różniczkowania:

$\int x^2\,dx=(x^3/3)+C$ gdzie C to stała całkowania.

- **Całka oznaczona**: Mierzy pole pod wykresem funkcji f(x) na przedziale [a,b]:

$$\int_a^b f(x)\,dx$$

Przykład:
Pole pod $f(x)=x^2$ na [0,2]:

$$\int_0^2 x^2\,dx = \left[\frac{x^3}{3}\right]_0^2 = \frac{2^3}{3} - \frac{0^3}{3} = \frac{8}{3}$$

2. Zastosowania pochodnych i całek

Zastosowania pochodnych

- **Fizyka**: Prędkość v(t) i przyspieszenie a(t) są odpowiednio pierwszą i drugą pochodną położenia s(t).
- **Ekonomia**: Maksymalizacja zysków lub minimalizacja kosztów, np. badanie funkcji kosztu C(x) lub przychodów R(x).
- **Inżynieria**: Analiza naprężeń i sił w konstrukcjach, gdzie pochodna opisuje zmiany sił w zależności od odległości.
- **Ruch prostoliniowy**:
  Jeśli położenie ciała wyraża się funkcją $s(t)=5t^2$, to prędkość v(t) jest pierwszą pochodną: $v(t) = s'(t) = 10t$
  przyspieszenie to druga pochodna:
  $a(t)=v'(t)=s''(t)=10$
- **Ekstremum funkcji w ekonomii**:
  Funkcja kosztu $C(q)=q^3-6q^2+9q$ opisuje koszty produkcji q. Znajdź punkt minimalnego kosztu: $C'(q)=3q^2-12q+9$

Druga pochodna $C''(q)=6q-12$ pozwala określić, że minimum występuje przy q=1.

Zastosowania całek

- **Obliczanie pól i objętości**: Całki pozwalają obliczać pole pod wykresem funkcji i objętość brył obrotowych.
- **Przepływ**: Całki obliczają całkowity przepływ przez dany obszar w hydrodynamice.
- **Fizyka**: Obliczanie pracy wykonanej przez siłę zmienną F(x) na drodze [a,b]:

$W=\int_a^b F(x)\,dx$

Przykład:
Objętość bryły powstałej przez obrót $y=x^2$ wokół osi x na [0,1]:

$$V = \pi \int_0^1 (x^2)^2 \, dx = \pi \int_0^1 x^4 \, dx = \pi \cdot \frac{1}{5} = \frac{\pi}{5}.$$

3. Liczby zespolone

Definicja

Liczby zespolone to liczby postaci z = a+bi, gdzie a to część rzeczywista, b to część urojona, a i spełnia $i^2 = -1$

Operacje na liczbach zespolonych

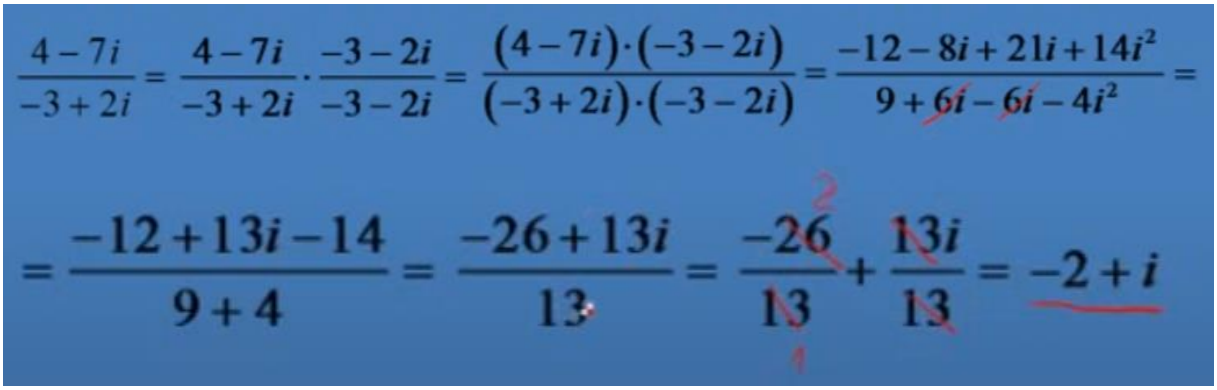
- **Dodawanie**:

(2+3i)+(1−i)=3+2i
Odejmowanie na tej samej zasadzie

- **Mnożenie**:

$(1+2i)(3+i)=3+i+6i+2i^2=3+7i+(2*(-1))=1+7i$

Dzielenie



Moduł i sprzężenie

- Moduł liczby z=a+bi:

$$|z| = \sqrt{a^2 + b^2}$$

Sprzężenie liczby z=a+bi:
$\bar{z}$ = a−bi

Zastosowanie w równaniach

Równanie kwadratowe $x^2+1=0$ ma rozwiązania zespolone: x=±i

4. Macierze i wyznaczniki. Zastosowania do rozwiązania układu równań liniowych

Macierze

Macierz to tablica liczb uporządkowana w wiersze i kolumny, np.:

$$A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$$

Operacje na macierzach

- **Dodawanie**:
  Dodajemy elementy o tych samych indeksach.
- **Mnożenie**:
  Macierz A·B istnieje, gdy liczba kolumn A równa się liczbie wierszy B.

Wyznacznik

Wyznacznik macierzy

$$A = \begin{vmatrix} a & b \\ c & d \end{vmatrix}$$

$$\det(A) = ad - bc$$

Rozwiązywanie układów równań

Układ równań:

{x+2y=5
{3x+4y=6.

W formie macierzowej:

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} * \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 5 \\ 6 \end{bmatrix}$$

Rozwiązanie:

- Wyznaczamy det(A)
- Obliczamy macierz odwrotną A^{-1}
- Rozwiązanie: $\begin{bmatrix} x \\ y \end{bmatrix} = A^{-1} \cdot \begin{bmatrix} 5 \\ 6 \end{bmatrix}$

1. Omówić własność bijektywności funkcji.
2. Reguła de l'Hospitala.
3. Warunki istnienia ekstremum lokalnego dla funkcji jednej zmiennej rzeczywistej.
4. Wzór Taylora i jego zastosowania.

5. Wyjaśnić pojęcie całki nieoznaczonej sposoby całkowania.
6. Omówić kryteria zbieżności (Cauchy' ego, d'Alamberta, porównawcze) dla szeregów o wyrazach dodatnich.
7. Wyjaśnić związek między całką oznaczoną a polem pod wykresem funkcji
8. Reprezentacje liczb zespolonych.
9. Zapis macierzowy układu równań liniowych i konsekwencje w postaci rozwiązań.

1. Własność bijektywności funkcji

Funkcja f(x) jest **bijektywna**, jeśli jest zarówno **iniektywna** (różnowartościowa), jak i **suriektywna** (na).

- **Iniektywność**: Funkcja jest iniektywna, gdy różnym argumentom przyporządkowuje różne wartości. Formalnie:

  $f(x_1)=f(x_2) \implies x_1=x_2$

- **Suriektywność**: Funkcja jest suriektywna, gdy każdy element przeciwdziedziny Y jest obrazem co najmniej jednego elementu z dziedziny X.
- **Bijekcja**: Funkcja jest bijektywna, jeśli dla każdego $y \in Y$ istnieje dokładnie jedno $x \in X$ takie, że $f(x)=y$. Tylko funkcje bijektywne mają funkcję odwrotną.

Przykład:
Funkcja liniowa $f(x)=2x+3$ jest bijektywna, bo jest różnowartościowa i pokrywa całą przeciwdziedzinę.

2. Reguła de l'Hospitala

Reguła de l'Hospitala jest metodą obliczania granic funkcji w sytuacjach nieoznaczonych 0/0 lub ∞/∞ Jeśli:

$$\lim_{x \to a} f(x) = 0 \text{ i } \lim_{x \to a} g(x) = 0 \quad \text{lub} \quad \lim_{x \to a} f(x) = \infty \text{ i } \lim_{x \to a} g(x) = \infty,$$

to:

$$\lim_{x \to a} \frac{f(x)}{g(x)} = \lim_{x \to a} \frac{f'(x)}{g'(x)},$$

o ile granica po prawej stronie istnieje.

Przykład:

$$\lim_{x\to 0}\frac{\sin(x)}{x}=\lim_{x\to 0}\frac{\cos(x)}{1}=1.$$

3. Warunki istnienia ekstremum lokalnego dla funkcji jednej zmiennej rzeczywistej

Definicja ekstremum lokalnego

Punkt x_0 jest ekstremum lokalnym funkcji f(x), jeśli:

- $f(x_0)$ jest wartością maksymalną lub minimalną w pewnym otoczeniu punktu x_0.

Warunki istnienia ekstremum lokalnego

1. **Warunek pierwszej pochodnej**:
   Jeśli x_0 jest ekstremum lokalnym, to $f'(x_0)=0$ lub $f'(x_0)$ nie istnieje.
2. **Warunek drugiej pochodnej**:
   Jeśli $f'(x_0)=0$ i $f''(x_0)>0$, to x_0 jest minimum lokalnym.
   Jeśli $f'(x_0)=0$ i $f''(x_0)<0$, to x_0 jest maksimum lokalnym.

Przykład:
Dla $f(x)=x^3-3x^2+4$ punkt x=2 jest minimum lokalnym, bo $f'(x)=0$ i $f''(x)>0$.

4. Wzór Taylora i jego zastosowania

Wzór Taylora pozwala przybliżyć funkcję f(x) za pomocą wielomianu wokół punktu x_0:

$$f(x)\approx f(x_0)+f'(x_0)(x-x_0)+\frac{f''(x_0)}{2!}(x-x_0)^2+\cdots+\frac{f^{(n)}(x_0)}{n!}(x-x_0)^n.$$

Zastosowania wzoru Taylora:

- Przybliżenie funkcji trudnych do analizy.
- Obliczenia numeryczne.
- Analiza błędów w przybliżeniach.

Przykład:
Rozwinięcie e^x w szereg Taylora wokół $x_0=0$:

$$e^x\approx 1+x+\frac{x^2}{2!}+\frac{x^3}{3!}+\dots.$$

5. Pojęcie całki nieoznaczonej i sposoby całkowania

Całka nieoznaczona

Całka nieoznaczona funkcji f(x)f(x)f(x) to zbiór wszystkich funkcji pierwotnych, których pochodna jest równa f(x):

$$\int f(x)\,dx = F(x) + C, \quad \text{gdzie } F'(x) = f(x) \text{ i } C \text{ to stała całkowania.}$$

Sposoby całkowania

1. **Całkowanie podstawowe**:

$$\int x^n\,dx = \frac{x^{n+1}}{n+1} + C \quad \text{dla } n \neq -1.$$

2. **Metoda podstawienia**: Używana, gdy można uprościć funkcję za pomocą zmiennej pomocniczej.
3. **Metoda przez części**: Stosowana do iloczynu dwóch funkcji:

$$\int u\,dv = uv - \int v\,du.$$

6. Kryteria zbieżności szeregów o wyrazach dodatnich

Kryterium Cauchy'ego

Szereg $\sum a_n$ o wyrazach dodatnich jest zbieżny, jeśli:

$$\lim_{n\to\infty} \sqrt[n]{a_n} < 1.$$

Kryterium d'Alemberta

Szereg $\sum a_n$ jest zbieżny, jeśli:

$$\lim_{n\to\infty} \frac{a_{n+1}}{a_n} < 1.$$

Kryterium porównawcze

Jeśli $0 \leq a_n \leq b_n$ to:

- Jeśli $\sum b_n$ jest zbieżny, to $\sum a_n$ też jest zbieżny.
- Jeśli $\sum a_n$ jest rozbieżny, to $\sum b_n$ też jest rozbieżny.

7. Związek między całką oznaczoną a polem pod wykresem funkcji

Całka oznaczona funkcji f(x) na przedziale [a,b]:

$\int_a^b f(x)\,dx$

mierzy **algebraiczne pole** pod wykresem funkcji:

- Obszary nad osią x mają pole dodatnie.
- Obszary pod osią x mają pole ujemne.

Przykład:
Dla $f(x)=x^2$ na [0,2]:

$\int_0^2 x^2\,dx = 2^3/3 - 0^3/3 = 8/3$

8. Reprezentacje liczb zespolonych

Liczby zespolone z=a+bi można reprezentować:

1. **Postać algebraiczna**: z=a+bi, gdzie a to część rzeczywista, a b to część urojona.
2. **Postać trygonometryczna**:

$Z = r(\cos\theta + i\sin\theta)$

gdzie $r=|z|$ (moduł) , $\theta=\arg(z)$ (argument)

3. **Postać wykładnicza**:

$z=re^{i\theta}$

gdzie $e^{i\theta}=\cos\theta+i\sin\theta$ (z równań Eulera).

9. Zapis macierzowy układu równań liniowych i konsekwencje

Zapis macierzowy

Układ równań:

$$\begin{cases} x+2y=5 \\ 3x+4y=6. \end{cases}$$

W zapisie macierzowej:

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} * \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 5 \\ 6 \end{bmatrix}$$

Konsekwencje

- **Rozwiązywalność**:
  Układ ma rozwiązanie, jeśli wyznacznik macierzy głównej jest różny od zera (det≠0).
- **Macierz odwrotna**:
  Jeśli A jest macierzą główną, to: $x=A^{-1}b$

Programowanie Obiektowe

1. Pojęcie klasy i obiektu na przykładzie języka C++

- **Klasa** to szablon, który definiuje strukturę i zachowanie obiektów. Zawiera **atrybuty** (zmienne) i **metody** (funkcje).
- **Obiekt** to konkretna instancja klasy, która reprezentuje rzeczywisty byt w programie.

Przykład w C++:

```
#include <iostream>
using namespace std;

class Samochod {
private:
   string marka;
   int rocznik;

public:
   // Konstruktor
   Samochod(string m, int r) : marka(m), rocznik(r) {}

   // Metoda do wyświetlania informacji o samochodzie
   void pokazInformacje() {
      cout << "Marka: " << marka << ", Rocznik: " << rocznik << endl;
   }
};

int main() {
   // Tworzenie obiektu klasy Samochod
   Samochod samochod1("Toyota", 2020);
   samochod1.pokazInformacje();

   return 0;
}
```

- Klasa Samochod definiuje atrybuty marka i rocznik oraz metodę pokazInformacje.
- Obiekt samochod1 jest instancją klasy Samochod.

2. Mechanizm dziedziczenia klas i polimorfizm w C++. Koncepcja hermetyzacji (enkapsulacji).

Dziedziczenie

Dziedziczenie umożliwia tworzenie nowych klas (klas pochodnych) na podstawie już istniejących (klas bazowych). Klasy pochodne dziedziczą atrybuty i metody klasy bazowej.

```
class Pojazd {
public:
   void uruchomSilnik() {
      cout << "Silnik uruchomiony." << endl;
   }
};

class Samochod : public Pojazd {
public:
   void uruchomRadio() {
      cout << "Radio uruchomione." << endl;
   }
};

int main() {
   Samochod auto1;
   auto1.uruchomSilnik();
   auto1.uruchomRadio();

   return 0;}
```

Polimorfizm

Polimorfizm pozwala na wywoływanie metod klas pochodnych przez wskaźniki/referencje do klasy bazowej.

```
class Pojazd {
public:
   virtual void uruchom() {
      cout << "Pojazd uruchomiony." << endl;
   }
};

class Rower : public Pojazd {
public:
   void uruchom() override {
      cout << "Rower gotowy do jazdy." << endl;
   }
};

int main() {
   Pojazd* p = new Rower();
   p->uruchom(); // Wywołuje metodę Rower::uruchom
   delete p;

   return 0;}
```

Hermetyzacja (enkapsulacja)

Hermetyzacja polega na ukrywaniu implementacji klasy (np. poprzez prywatne atrybuty) i udostępnianiu tylko niezbędnych funkcji publicznych.

3. Klasy abstrakcyjne w języku C++

Klasa abstrakcyjna to klasa, która zawiera co najmniej jedną metodę wirtualną czysto abstrakcyjną (tzw. "pure virtual function"). Nie można utworzyć obiektu klasy abstrakcyjnej.

```
class Figura {
public:
   virtual void rysuj() = 0; // Metoda abstrakcyjna
};

class Kwadrat : public Figura {
public:
   void rysuj() override {
      cout << "Rysowanie kwadratu." << endl;
   }
};

int main() {
   Kwadrat k;
   k.rysuj();

   return 0;
}
```

- Metoda rysuj musi być zaimplementowana w klasie pochodnej.

4. Wskaźniki i referencje oraz dynamiczne zarządzanie pamięcią

Wskaźniki

Wskaźnik przechowuje adres pamięci innej zmiennej.

```
int x = 10;
int* p = &x; // Wskaźnik na x
cout << *p << endl; // Dereferencja wskaźnika
```

Referencje

Referencja jest aliasem dla istniejącej zmiennej.

```
int x = 10;
int& ref = x; // Referencja do x
ref = 20; // Zmiana wartości x
```

Dynamiczne zarządzanie pamięcią

W języku C++ używa się operatorów new i delete.

```
int* p = new int(5); // Dynamiczna alokacja pamięci
cout << *p << endl;
delete p; // Zwolnienie pamięci
```

5. Przeciążenie operatorów, funkcji oraz koncepcja funkcji zaprzyjaźnionych

Przeciążenie operatorów

Operator może być przeciążony, aby działał na obiektach klasy.

```
class Liczba {
private:
   int wartosc;

public:
   Liczba(int w) : wartosc(w) {}

   Liczba operator+(const Liczba& druga) {
      return Liczba(wartosc + druga.wartosc);
   }

   void pokaz() {
      cout << wartosc << endl;
   }
};

int main() {
   Liczba a(5), b(10);
   Liczba c = a + b;
   c.pokaz();

   return 0;
}
```

Funkcje zaprzyjaźnione

Funkcje zaprzyjaźnione mają dostęp do prywatnych składników klasy.

```
class Klasa {
private:
   int dane;

public:
   Klasa(int d) : dane(d) {}

   friend void pokaz(const Klasa& k);
};

void pokaz(const Klasa& k) {
   cout << "Dane: " << k.dane << endl;}
```

6. Poziomy dostępu do składników klasy oraz składniki statyczne

- **Publiczne**: Dostępne dla wszystkich.
- **Chronione (protected)**: Dostępne dla klasy i jej dziedziczących.
- **Prywatne**: Dostępne tylko wewnątrz klasy.

Składniki statyczne

- Są współdzielone między wszystkimi obiektami klasy.
- Należą do klasy, a nie do konkretnego obiektu.

```
class Klasa {
public:
   static int licznik;

   Klasa() {
      licznik++;
   }
};

int Klasa::licznik = 0;

int main() {
   Klasa ob1, ob2;
   cout << Klasa::licznik << endl;

   return 0;
}
```

7. Konstruktory i destruktory w C++

Konstruktor

- Automatycznie wywoływany przy tworzeniu obiektu.
- Inicjalizuje obiekt.

Destruktor

- Automatycznie wywoływany przy usuwaniu obiektu.
- Zwalnia zasoby.

```
class Klasa {
public:
   Klasa() {
      cout << "Konstruktor" << endl;
   }

   ~Klasa() {
      cout << "Destruktor" << endl;
   }
};
```

8. Obsługa wyjątków i biblioteka STL

Obsługa wyjątków

Mechanizm zarządzania błędami.

```
try {
   throw runtime_error("Błąd!");
} catch (const runtime_error& e) {
   cout << "Wyjątek: " << e.what() << endl;
}
```

Standardowa Biblioteka Szablonów (STL)

STL dostarcza struktury danych i algorytmy:

- **Kontenery**: np. vector, list, map.
- **Iteratory**: pozwalają na przeglądanie elementów.
- **Algorytmy**: np. sort, find.

```
#include <vector>
#include <algorithm>

vector<int> liczby = {3, 1, 4};
sort(liczby.begin(), liczby.end());
```

1. Mechanizm dziedziczenia klas – idea, zastosowania.
2. Klasy abstrakcyjne – idea, zastosowania.
3. Polimorfizm – idea, rodzaje, zastosowania.
4. Pojęcie klasy i obiektu.
5. Podstawowe techniki programowania.
6. Poziomy dostępu do składników klasy w języku C++.
7. Konstruktory i destruktory w C++ - omówienie.
8. Koncepcja hermetyzacji (enkapsulacji) w programowaniu obiektowym.
9. Biblioteka STL – krótkie omówienie, zawartość, przykłady kontenerów.

1. Mechanizm dziedziczenia klas – idea, zastosowania

Idea

Dziedziczenie to mechanizm umożliwiający tworzenie nowych klas (klas pochodnych) na podstawie już istniejących (klas bazowych). Klasa pochodna dziedziczy składniki klasy bazowej (atrybuty i metody), co pozwala na wielokrotne wykorzystanie kodu oraz jego rozbudowę.

Zastosowania

- **Kod wielokrotnego użytku**: Klasy bazowe definiują wspólne cechy, a klasy pochodne dodają specyficzne funkcjonalności.
- **Hierarchie klas**: Organizowanie powiązań między obiektami.
- **Rozszerzanie funkcjonalności**: Dodawanie nowych metod i atrybutów do istniejących klas.

Przykład

```
class Pojazd {
public:
   void uruchomSilnik() {
      cout << "Silnik uruchomiony." << endl;
   }
};

class Samochod : public Pojazd {
public:
   void uruchomRadio() {
      cout << "Radio uruchomione." << endl;
   }
};
```

2. Klasy abstrakcyjne – idea, zastosowania

Idea

Klasa abstrakcyjna to klasa, która zawiera co najmniej jedną metodę czysto wirtualną (`= 0`). Nie można tworzyć jej instancji, służy jako szablon dla klas pochodnych.

Zastosowania

- **Wymuszanie implementacji**: Klasa abstrakcyjna wymusza implementację określonych metod w klasach pochodnych.
- **Polimorfizm**: Umożliwia wywoływanie metod za pomocą wskaźnika do klasy bazowej.
- **Tworzenie interfejsów**: Definiowanie zbioru metod do zaimplementowania przez różne klasy.

Przykład

```
class Figura {
public:
   virtual void rysuj() = 0; // Metoda abstrakcyjna
};

class Kwadrat : public Figura {
public:
   void rysuj() override {
      cout << "Rysowanie kwadratu." << endl;
   }
};
```

3. Polimorfizm – idea, rodzaje, zastosowania

Idea

Polimorfizm pozwala na różne zachowanie metod, w zależności od obiektu, na którym są wywoływane, nawet jeśli obiekt jest używany przez wskaźnik do klasy bazowej.

Rodzaje

1. **Polimorfizm statyczny** (kompilacyjny):
    - Przeciążanie funkcji i operatorów.
    - Decyzje podejmowane w czasie kompilacji.
2. **Polimorfizm dynamiczny** (w czasie wykonania):
    - Użycie funkcji wirtualnych.
    - Decyzje podejmowane w czasie działania programu.

Zastosowania

- **Interfejsy**: Ujednolicanie sposobu korzystania z różnych obiektów.
- **Elastyczność**: Kod działa na klasach pochodnych bez konieczności ich znajomości w czasie kompilacji.

Przykład

```
class Pojazd {
public:
   virtual void uruchom() {
      cout << "Uruchamianie pojazdu." << endl;
   }
};

class Rower : public Pojazd {
public:
   void uruchom() override {
      cout << "Rower gotowy do jazdy." << endl;
   }
};
```

4. Pojęcie klasy i obiektu

- **Klasa**: Szablon definiujący strukturę i zachowanie obiektów. Zawiera dane (atrybuty) i funkcje (metody).
- **Obiekt**: Instancja klasy, reprezentująca konkretny byt w programie.

Przykład

```
class Samochod {
   string marka;
   int rocznik;
};
Samochod samochod1; // Obiekt klasy Samochod
```

5. Podstawowe techniki programowania

1. Programowanie proceduralne

Opis:

- Jest to podejście, w którym program dzieli się na procedury (funkcje), które operują na danych.
- Dane i funkcje są oddzielone; funkcje przyjmują dane jako argumenty, przetwarzają je i zwracają wyniki.

Kluczowe cechy:

- **Podział na funkcje:** Każda funkcja wykonuje konkretną część zadania, np. obliczenia, przetwarzanie danych, interfejs użytkownika.
- **Hierarchia wywołań:** Główna funkcja steruje przepływem programu, wywołując inne funkcje.
- **Łatwość zrozumienia:** Dzięki podziałowi na mniejsze części kod staje się bardziej przejrzysty.

Zalety:

- Łatwość implementacji i debugowania.
- Dobre dla małych i średnich projektów.
- Intuicyjne podejście dla początkujących.

Wady:

- Trudności w zarządzaniu danymi w dużych projektach.
- Brak enkapsulacji danych (dane mogą być dostępne i modyfikowane przez wiele funkcji).

Przykład zastosowań:

- Skrypty automatyzujące.
- Programy systemowe, np. narzędzia konsolowe.

2. Programowanie obiektowe (OOP - Object-Oriented Programming)

Opis:

- Kod jest organizowany wokół obiektów, które łączą dane i funkcje w jedną jednostkę.
- Każdy obiekt jest instancją klasy, która definiuje jego strukturę i zachowanie.

Kluczowe cechy:

1. **Enkapsulacja:** Dane (pola) i metody (funkcje) są zintegrowane w jednej klasie.

2. **Dziedziczenie:** Klasy mogą dziedziczyć cechy i funkcjonalność innych klas, co umożliwia ponowne wykorzystanie kodu.
3. **Polimorfizm:** Funkcje i obiekty mogą przyjmować różne formy w zależności od kontekstu.
4. **Abstrakcja:** Skupienie na istotnych szczegółach i ukrywanie zbędnych informacji.

Zalety:

- Łatwość ponownego użycia kodu (reusability).
- Lepsze zarządzanie złożonością dużych projektów.
- Naturalne modelowanie rzeczywistości (np. obiekty w grze: gracz, przeciwnik, przedmioty).

Wady:

- Większa złożoność początkowa w porównaniu z programowaniem proceduralnym.
- Nie zawsze jest konieczne (nadmiarowość w małych projektach).

Przykład zastosowań:

- Aplikacje desktopowe i mobilne.
- Gry komputerowe.
- Systemy zarządzania relacjami z klientami (CRM).

3. Programowanie funkcyjne

Opis:

- Opiera się na używaniu funkcji jako podstawowego budulca programów.
- Funkcje w programowaniu funkcyjnym są "czyste" — ich wynik zależy tylko od wejścia, bez efektów ubocznych.

Kluczowe cechy:

1. **Czystość funkcji:** Brak efektów ubocznych (funkcja nie zmienia zewnętrznych danych).
2. **Immutability:** Dane są niezmienne (każda operacja zwraca nową kopię danych zamiast modyfikować istniejące).
3. **Wyrażenia:** Każda część kodu jest wyrażeniem i zwraca wartość.
4. **Rekurencja:** Rekurencja zastępuje pętle iteracyjne.

Zalety:

- Łatwość testowania i debugowania dzięki braku efektów ubocznych.
- Lepsza współbieżność (concurrency) dzięki niezmienności danych.
- Zwięzłość i elegancja kodu.

Wady:

- Może być trudniejsze do zrozumienia dla osób przyzwyczajonych do programowania proceduralnego.
- Rekurencja bywa mniej wydajna niż iteracja.

Przykład zastosowań:

- Przetwarzanie danych (np. w big data, map-reduce).
- Aplikacje przetwarzające strumienie danych.
- Programowanie współbieżne i równoległe.

4. Programowanie generyczne

Opis:

- Polega na pisaniu kodu, który może działać z różnymi typami danych bez konieczności ich specyfikowania na etapie implementacji.
- W C++ realizowane za pomocą szablonów (templates).

Kluczowe cechy:

1. **Abstrakcja typów:** Szablony umożliwiają definiowanie funkcji i klas, które są niezależne od konkretnego typu danych.
2. **Dynamiczne dostosowanie:** Typy są określane w momencie użycia, a nie definicji.

Zalety:

- Uniwersalność: Jeden kod może działać na różnych typach danych.
- Reusability: Możliwość ponownego użycia kodu bez potrzeby przepisywania go dla każdego typu.
- Typowanie statyczne: Błędy są wykrywane na etapie kompilacji.

Wady:

- Kod generyczny może być trudny do zrozumienia i debugowania.
- Może prowadzić do większej liczby wygenerowanego kodu, co zwiększa rozmiar binarny aplikacji.

Przykład zastosowań:

- Kontenery i algorytmy w bibliotekach standardowych (np. STL w C++).
- Frameworki i biblioteki do obsługi różnych typów danych.

Porównanie technik:

Technika	Zalety	Wady
Proceduralne	Proste w implementacji, intuicyjne.	Trudne w utrzymaniu w dużych projektach.
Obiektowe	Skalowalność, ponowne użycie kodu.	Większa złożoność.
Funkcyjne	Łatwość debugowania, bez efektów ubocznych.	Trudniejsze dla początkujących.
Generyczne	Uniwersalność, elastyczność.	Trudności w debugowaniu.

6. Poziomy dostępu do składników klasy w języku C++

1. **Publiczne** (public):
   - Dostępne dla wszystkich funkcji i klas.
2. **Chronione** (protected):
   - Dostępne w klasie oraz w jej klasach pochodnych.
3. **Prywatne** (private):
   - Dostępne tylko wewnątrz klasy.

7. Konstruktory i destruktory w C++ - omówienie

- **Konstruktor**: Automatycznie wywoływany przy tworzeniu obiektu, inicjalizuje dane.
- **Destruktor**: Automatycznie wywoływany przy niszczeniu obiektu, zwalnia zasoby.

Przykład

```
class Klasa {
public:
   Klasa() { cout << "Konstruktor." << endl; }
   ~Klasa() { cout << "Destruktor." << endl; }};
```

8. Koncepcja hermetyzacji (enkapsulacji) w programowaniu obiektowym

Hermetyzacja polega na ukrywaniu szczegółów implementacji klasy i udostępnianiu jedynie niezbędnych funkcji publicznych. Ułatwia to zarządzanie kodem i poprawia bezpieczeństwo.
Przykład

```
class KontoBankowe {
private:
   double saldo;

public:
   void wpłać(double kwota) { saldo += kwota; }
   double sprawdzSaldo() { return saldo; }
};
```

9. Biblioteka STL – krótkie omówienie, zawartość, przykłady kontenerów

Krótkie omówienie

STL (Standard Template Library) to zestaw gotowych komponentów do pracy z danymi. Zawiera:

- **Kontenery**: Przechowywanie danych.
- **Iteratory**: Dostęp do elementów kontenera.
- **Algorytmy**: Operacje na danych, np. sortowanie.

Przykłady kontenerów

1. **Vector**: Dynamiczna tablica.
2. **List**: Lista dwukierunkowa.
3. **Map**: Tablica asocjacyjna (klucz-wartość).
4. **Set**: Zbiór unikalnych elementów.

Przykład

```
#include <vector>
#include <iostream>
using namespace std;

int main() {
   vector<int> liczby = {3, 1, 4};
   liczby.push_back(5);
   for (int liczba : liczby) {
      cout << liczba << " ";
   }
   return 0;
}
```

Systemy Operacyjne

1. Struktury systemów operacyjnych

Definicja

System operacyjny to oprogramowanie pośredniczące między sprzętem a użytkownikiem, umożliwiające zarządzanie zasobami komputera.

Struktury systemów operacyjnych

1. **Jednopoziomowe (monolityczne)**:
    - Cały kod systemu operacyjnego działa jako jeden moduł w przestrzeni jądra.
    - Zaleta: Szybkość działania.
    - Wada: Trudność w zarządzaniu i testowaniu.

2. **Warstwowe**:
    - System podzielony jest na warstwy (np. sprzęt, zarządzanie procesami, system plików).
    - Zaleta: Modułowość i łatwość modyfikacji.
    - Wada: Dodatkowe narzuty wydajnościowe.
3. **Z mikrojądrem**:
    - Podstawowe funkcje, takie jak komunikacja między procesami, są realizowane przez mikrojądro, a reszta funkcji działa w przestrzeni użytkownika.
    - Zaleta: Bezpieczeństwo i niezawodność.
    - Wada: Mniejsza wydajność niż w przypadku systemów monolitycznych.
4. **Systemy hybrydowe**:
    - Połączenie mikrojądra i jądra monolitycznego.
    - Popularne w nowoczesnych systemach operacyjnych (np. Windows, macOS).

2. Zarządzanie procesami

Proces

Proces to działający program, który składa się z kodu, danych, sterty (heap) i stosu (stack). Procesy są zarządzane przez system operacyjny.

Zarządzanie procesami obejmuje:

1. **Tworzenie i usuwanie procesów**:
    - Procesy są tworzone przez system operacyjny lub inne procesy (np. proces nadrzędny tworzy proces podrzędny).
2. **Przydział CPU (planowanie procesora)**:
    - Określenie, który proces otrzyma dostęp do procesora. Algorytmy planowania:
        - FCFS (First Come, First Served).
        - SJF (Shortest Job First).
        - Round Robin.
3. **Stan procesów**:
    - Nowy, gotowy, wykonywany, oczekujący, zakończony.
4. **Przełączanie kontekstu**:
    - Przełączanie procesora między procesami, aby umożliwić wielozadaniowość.

3. Synchronizacja procesów

Definicja

Synchronizacja procesów jest kluczowa w systemach wielozadaniowych, gdzie procesy mogą współdzielić zasoby i wymagać koordynacji.

Problemy i mechanizmy synchronizacji:

1. **Wyścig danych**:
    - Dwa procesy próbują jednocześnie modyfikować ten sam zasób.
2. **Sekcje krytyczne**:
    - Fragmenty kodu, które mogą być wykonywane tylko przez jeden proces na raz.
3. **Mechanizmy synchronizacji**:
    - **Muteksy**: Blokują dostęp do zasobu dla jednego procesu.
    - **Semafory**: Licznik pozwalający kontrolować dostęp do zasobów.
    - **Monitory**: Abstrakcje synchronizujące dostęp do zasobów.
4. **Problemy synchronizacji**:
    - Problem producenta-konsumenta.
    - Problem filozofów przy stole.

4. Zarządzanie zasobami pamięci operacyjnej

Podział pamięci operacyjnej

1. **Pamięć fizyczna**: Rzeczywista pamięć RAM komputera.
2. **Pamięć wirtualna**: Abstrakcja pamięci fizycznej, umożliwia korzystanie z większej przestrzeni pamięciowej niż dostępna RAM.

Techniki zarządzania pamięcią

1. **Podział na partycje**:
    - Stały: Równe rozmiary bloków.
    - Dynamiczny: Bloki o zmiennym rozmiarze.
2. **Stronicowanie**:
    - Pamięć dzielona na równe bloki (strony).
    - Rozwiązuje problem fragmentacji zewnętrznej.
3. **Segmentacja**:
    - Pamięć dzielona według logicznych segmentów (np. kod, dane).
4. **Algorytmy zarządzania pamięcią**:
    - FIFO (First In, First Out).
    - LRU (Least Recently Used).

5. Zarządzanie pamięcią masową

Pamięć masowa

Pamięć masowa to urządzenia przechowujące dane przez dłuższy czas, np. dyski twarde, SSD.

Zarządzanie pamięcią masową obejmuje:

1. **Planowanie dostępu do dysku**:
    - Algorytmy, takie jak FCFS, SSTF (Shortest Seek Time First), SCAN.
2. **Buforowanie**:

    - Przechowywanie często używanych danych w szybszej pamięci (cache).
3. **RAID (Redundant Array of Independent Disks)**:
    - Technologia zwiększająca wydajność i niezawodność pamięci masowej.

6. System plików

System plików zarządza przechowywaniem i organizacją plików na urządzeniach masowych.

Elementy systemu plików:

1. **Struktura katalogów**:
    - Hierarchiczna organizacja plików.
    - Struktura może być jednopoziomowa, wielopoziomowa, z możliwością symbolicznych odnośników.
2. **Metadane**:
    - Informacje o pliku, np. rozmiar, data utworzenia, prawa dostępu.
3. **Operacje na plikach**:
    - Tworzenie, odczyt, zapis, usuwanie.
4. **Systemy plików**:
    - FAT, NTFS, ext3/ext4.

7. Bezpieczeństwo i ochrona w systemach operacyjnych

Ochrona to zabezpieczenie zasobów systemu przed nieautoryzowanym dostępem, a bezpieczeństwo to zabezpieczenie danych przed utratą i uszkodzeniem.

Zagadnienia bezpieczeństwa i ochrony:

1. **Mechanizmy kontroli dostępu**:
    - Uwierzytelnianie użytkownika (hasła, klucze).
    - Listy kontroli dostępu (ACL).
2. **Ochrona procesów**:
    - Mechanizmy izolujące procesy od siebie.
3. **Szyfrowanie danych**:
    - Chroni dane przechowywane i przesyłane w systemie.
4. **Zabezpieczenia przed złośliwym oprogramowaniem**:
    - Zapory ogniowe, oprogramowanie antywirusowe.
5. **Audyt systemu**:
    - Monitorowanie aktywności w celu wykrycia naruszeń bezpieczeństwa.

1. Jakie zasoby są używane podczas tworzenia wątku? Czym różnią się od tych używanych podczas tworzenia procesu?

2. Zdefiniuj różnicę między planowaniem z wywłaszczaniem a planowaniem niewywłaszczającym.

3. Wymień trzy przykłady zakleszczeń, które nie są związane ze środowiskiem systemu komputerowego.

4. W jakich okolicznościach występują błędy strony? Opisz działania podjęte przez system operacyjny, gdy wystąpi błąd strony.

5. Wyjaśnij, dlaczego planowanie SSTF ma tendencję do faworyzowania środkowych cylindrów w porównaniu z najbardziej wewnętrznymi i zewnętrznymi cylindrami.
6. W jaki sposób DMA zwiększa współbieżność systemu? Jak to komplikuje projektowanie sprzętu?
7. Jakie są główne różnice między listami możliwości a listami dostępu?
8. Jakie są główne czynności systemu operacyjnego w odniesieniu do zarządzania procesami?
9. Wyjaśnij, pod względem kosztów, różnice między trzema typami pamięci masowej: ulotne, nieulotne i stabilne.
10. Wymień dwie różnice między adresami logicznymi i fizycznymi.
11. W jaki sposób pamięci podręczne pomagają poprawić wydajność? Dlaczego systemy nie używają większej lub większej liczby pamięci podręcznych, skoro są one tak przydatne?

1. Jakie zasoby są używane podczas tworzenia wątku? Czym różnią się od tych używanych podczas tworzenia procesu?

Tworzenie wątku

Wątek dzieli zasoby z procesem macierzystym, takie jak:

- Kod i dane programu.
- Segment sterty.
- Zasoby systemowe (np. uchwyty do plików).

Wątek wymaga:

- Oddzielnego stosu.
- Wskaźnika do rejestrów procesora.
- Własnego licznika programu.

Tworzenie procesu

Proces jest bardziej zasobożerny:

- Wymaga pełnej kopii segmentów kodu, danych i sterty (lub wskaźników do nich w przypadku mechanizmu kopiowania przy zapisie).
- Przydziela oddzielny obszar adresowy w pamięci.
- Rejestry procesora są niezależne.

Różnice

- Tworzenie wątku jest bardziej efektywne (niższe koszty pamięci i czasu).
- Wątki dzielą zasoby procesu, procesy mają własne, niezależne zasoby.

2. Zdefiniuj różnicę między planowaniem z wywłaszczaniem a planowaniem niewywłaszczającym.

- **Planowanie z wywłaszczaniem**:
  Procesor może być odebrany aktualnie działającemu procesowi przed jego zakończeniem, aby przydzielić go procesowi o wyższym priorytecie lub gdy upłynie limit czasu (np. Round Robin).
    - Zaleta: Lepsze wsparcie dla systemów czasu rzeczywistego.
    - Wada: Wyższy narzut na przełączanie kontekstu.
- **Planowanie niewywłaszczające**:
  Proces kontynuuje działanie, dopóki nie zakończy swojej pracy lub nie przejdzie w stan oczekiwania.
    - Zaleta: Prostsze i mniej kosztowne przełączanie.
    - Wada: Może prowadzić do głodzenia procesów o niższym priorytecie.

3. Wymień trzy przykłady zakleszczeń, które nie są związane ze środowiskiem systemu komputerowego.

1. **Ruch drogowy**: Skrzyżowanie, gdzie samochody z różnych kierunków blokują się wzajemnie, czekając na zwolnienie drogi.
2. **Gra planszowa**: Dwóch graczy czeka, aż przeciwnik wykona ruch, co prowadzi do impasu.
3. **Relacje handlowe**: Dwie firmy, które chcą współpracować, czekają, aż druga strona zrobi pierwszy krok.

4. W jakich okolicznościach występują błędy strony? Opisz działania podjęte przez system operacyjny, gdy wystąpi błąd strony.

Okoliczności

Błąd strony występuje, gdy proces próbuje uzyskać dostęp do strony, która nie jest obecnie załadowana w pamięci RAM, lecz znajduje się na dysku.

Działania systemu operacyjnego

1. Zatrzymanie procesu i sprawdzenie żądanej strony w tabeli stron.
2. Jeśli strona znajduje się na dysku:
    - Załaduj stronę do RAM-u (ewentualnie usuwając inną stronę w przypadku braku miejsca).
    - Zaktualizuj tabelę stron.
3. Wznowienie procesu.

5. Wyjaśnij, dlaczego planowanie SSTF ma tendencję do faworyzowania środkowych cylindrów w porównaniu z najbardziej wewnętrznymi i zewnętrznymi cylindrami.

SSTF (Shortest Seek Time First) wybiera żądanie zlokalizowane najbliżej aktualnej pozycji głowicy.

- Cylindry środkowe są częściej odwiedzane, ponieważ zarówno wewnętrzne, jak i zewnętrzne żądania muszą „przejść" przez środek.
- W wyniku tego żądania środkowe są obsługiwane szybciej, a te na krańcach mogą czekać dłużej (problem głodzenia).

6. W jaki sposób DMA zwiększa współbieżność systemu? Jak to komplikuje projektowanie sprzętu?

DMA (Direct Memory Access)

- Umożliwia urządzeniom peryferyjnym bezpośredni dostęp do pamięci RAM, omijając procesor.
- Zwiększa współbieżność, ponieważ procesor może wykonywać inne zadania podczas transferu danych przez DMA.

Komplikacje sprzętowe

- Wymaga dodatkowego kontrolera DMA.
- Konflikty w dostępie do pamięci między DMA a procesorem muszą być zarządzane.
- Złożoność projektowania magistrali systemowej.

7. Jakie są główne różnice między listami możliwości a listami dostępu?

1. **Listy możliwości (Capability Lists)**:
    - Przypisane do podmiotu (np. użytkownika lub procesu).
    - Opisują zasoby, do których dany podmiot ma dostęp.
    - Łatwiejsze zarządzanie uprawnieniami w kontekście użytkownika.
2. **Listy dostępu (Access Control Lists, ACL)**:
    - Przypisane do zasobu (np. pliku).
    - Opisują, którzy użytkownicy lub procesy mają dostęp do zasobu i na jakich zasadach.
    - Ułatwiają zarządzanie uprawnieniami dla zasobów.

8. Jakie są główne czynności systemu operacyjnego w odniesieniu do zarządzania procesami?

1. **Tworzenie i usuwanie procesów**.
2. **Planowanie procesora**: Decydowanie, który proces będzie wykonywany.
3. **Zarządzanie komunikacją międzyprocesową (IPC)**.
4. **Przełączanie kontekstu**: Obsługa zmiany procesu na proces.
5. **Obsługa stanu procesów**: Gotowy, wykonywany, oczekujący.
6. **Zarządzanie zasobami procesów**: Przydział pamięci, uchwyty plików.

9. Wyjaśnij, pod względem kosztów, różnice między trzema typami pamięci masowej: ulotne, nieulotne i stabilne.

1. **Ulotna pamięć (Volatile Memory)**:
    - Traci dane po wyłączeniu zasilania (np. RAM).
    - Szybka, ale droga.
2. **Nieulotna pamięć (Non-Volatile Memory)**:
    - Zachowuje dane po wyłączeniu zasilania (np. dyski SSD).
    - Wolniejsza od RAM, ale tańsza.
3. **Stabilna pamięć (Stable Storage)**:
    - Zabezpieczona przed awariami dzięki redundancji (np. RAID).
    - Kosztowna w implementacji i wolniejsza niż pamięć ulotna.

10. Wymień dwie różnice między adresami logicznymi i fizycznymi.

1. **Adres logiczny**:
    - Generowany przez procesor.
    - Odnosi się do wirtualnej przestrzeni adresowej procesu.
2. **Adres fizyczny**:
    - Odnosi się do rzeczywistego adresu w pamięci RAM.
    - Tłumaczony z adresu logicznego przez jednostkę zarządzania pamięcią (MMU).

11. W jaki sposób pamięci podręczne pomagają poprawić wydajność? Dlaczego systemy nie używają większej lub większej liczby pamięci podręcznych?

Rola pamięci podręcznej

- Pamięć podręczna przechowuje często używane dane w szybkim dostępie (blisko procesora).
- Redukuje czas oczekiwania na dane z pamięci głównej.

Dlaczego nie używamy większej liczby pamięci podręcznej?

1. **Koszt**: Pamięci podręczne są znacznie droższe niż pamięci główne.
2. **Wydajność**: Większa pamięć podręczna może obniżać szybkość ze względu na czas wyszukiwania.
3. **Złożoność**: Projektowanie bardziej złożonych hierarchii pamięci komplikuje system.

Przetwarzanie dokumentów XML i zaawansowane techniki WWW

1. XML, elementy, atrybuty

XML (eXtensible Markup Language)
XML to język znaczników używany do przechowywania i przenoszenia danych w formacie czytelnym zarówno dla ludzi, jak i maszyn. Jest uniwersalny, elastyczny i łatwy do rozbudowy.

Podstawowe składniki XML:

1. **Elementy XML**
   - Elementy to podstawowe jednostki strukturalne XML.
   - Zawierają dane lub inne elementy.
   - Składnia: <element>Treść elementu</element>
   - Mogą mieć elementy wewnętrzne (zagnieżdżone):

```
<parent>
    <child>Treść</child>
</parent>
```

   - Każdy element ma otwierający i zamykający znacznik (z wyjątkiem pustych elementów).
2. **Atrybuty XML**
   - Przechowują dodatkowe informacje o elemencie.
   - Definiowane w znaczniku otwierającym:

```
<element atrybut="wartość">Treść elementu</element>
```

   - Używane, gdy dane są metadanymi elementu.
3. **Deklaracja XML**
   - Na początku dokumentu określa wersję XML i kodowanie:

```
<?xml version="1.0" encoding="UTF-8"?>
```

Cechy XML:

- XML jest **case-sensitive** (wielkość liter ma znaczenie).
- Struktura musi być poprawna i hierarchiczna.
- Wartość atrybutów musi być w cudzysłowie.

2. Definicja struktury dokumentu za pomocą DTD (Document Type Definition)

DTD to mechanizm definiowania struktury i reguł dokumentu XML. Dzięki DTD można sprawdzić poprawność dokumentu XML pod kątem zgodności ze zdefiniowaną strukturą.

Rodzaje DTD:

1. **Wewnętrzna DTD**: Zawarta w samym pliku XML.
    - Przykład:

```
<?xml version="1.0"?>
<!DOCTYPE note [
  <!ELEMENT note (to,from,heading,body)>
  <!ELEMENT to (#PCDATA)>
  <!ELEMENT from (#PCDATA)>
  <!ELEMENT heading (#PCDATA)>
  <!ELEMENT body (#PCDATA)>
]>
<note>
  <to>Alice</to>
  <from>Bob</from>
  <heading>Reminder</heading>
  <body>Don't forget the meeting!</body>
</note>
```

2. **Zewnętrzna DTD**: Zapisywana w osobnym pliku i odwoływana w XML.
    - Przykład:

```
<?xml version="1.0"?>
<!DOCTYPE note SYSTEM "note.dtd">
<note>
  <to>Alice</to>
  <from>Bob</from>
  <heading>Reminder</heading>
  <body>Don't forget the meeting!</body>
</note>
```

    - Plik note.dtd:

```
<!ELEMENT note (to,from,heading,body)>
<!ELEMENT to (#PCDATA)>
<!ELEMENT from (#PCDATA)>
<!ELEMENT heading (#PCDATA)>
<!ELEMENT body (#PCDATA)>
```

Składnia DTD:

- **ELEMENT**: Definiuje strukturę elementów.
    - Przykład: <!ELEMENT nazwa (pod-elementy)>.
- **ATTLIST**: Definiuje atrybuty elementów.
    - Przykład: <!ATTLIST element nazwa_atrybutu TYPE DEFAULT>.

3. Renderowanie XML za pomocą CSS, transformacje XSLT

Renderowanie XML za pomocą CSS

XML można wyświetlać z użyciem stylów CSS, podobnie jak w przypadku HTML.

- XML definiuje dane, a CSS określa sposób ich prezentacji.

Przykład: XML:

```
<note>
  <to>Alice</to>
  <from>Bob</from>
  <heading>Reminder</heading>
  <body>Don't forget the meeting!</body>
</note>
```

CSS:

```
note {
  font-family: Arial, sans-serif;
  color: blue;
}

to {
  font-weight: bold;
}

from {
  font-style: italic;
}
```

Rezultat: XML wyświetlany w przeglądarce w sposób sformatowany.

Transformacje XSLT

XSLT (eXtensible Stylesheet Language Transformations) to język do przekształcania XML na inne formaty (np. HTML, tekst).

- Pozwala na dynamiczne generowanie dokumentów na podstawie XML.

Przykład: XML:

```
<note>
  <to>Alice</to>
  <from>Bob</from>
  <message>Hello, Alice!</message>
</note>
```

XSLT:

```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
  <xsl:template match="/">
    <html>
      <body>
        <h1>Message</h1>
        <p>To: <xsl:value-of select="note/to"/></p>
        <p>From: <xsl:value-of select="note/from"/></p>
        <p><xsl:value-of select="note/message"/></p>
```

```
        </body>
      </html>
    </xsl:template>
</xsl:stylesheet>
```

Rezultat: XML przekształcony w dokument HTML.

4. XPath i DOM

XPath (XML Path Language)

XPath (ang. *XML Path Language*) to język zapytań używany do nawigacji i wybierania węzłów w dokumentach XML. Umożliwia precyzyjne zapytania o dane w hierarchii XML.

Przykład: XML:

```
<bookstore>
  <book>
    <title>XML Basics</title>
    <price>29.99</price>
  </book>
  <book>
    <title>Advanced XML</title>
    <price>49.99</price>
  </book>
</bookstore>
```

XPath:

- /bookstore/book/title: Zwraca wszystkie tytuły książek.
- /bookstore/book[price>30]/title: Zwraca tytuły książek droższych niż 30.

DOM (Document Object Model)

DOM (Model Obiektowy Dokumentu) to standardowy interfejs programowania dla dokumentów HTML i XML. Reprezentuje strukturę dokumentu w postaci drzewa, gdzie każdy element, atrybut i tekst jest węzłem (*node*). DOM umożliwia programistom manipulowanie treścią, strukturą i stylem dokumentu za pomocą języków programowania, takich jak JavaScript.

Główne węzły DOM:

- **Element**: Każdy znacznik XML.
- **Atrybut**: Metadane elementu.
- **Tekst**: Zawartość elementów.

Cechy DOM

1. **Hierarchiczna struktura drzewa:**
    - Dokument jest przedstawiany jako drzewo, którego korzeniem jest węzeł document.

    - Węzły są połączone relacjami rodzic-dziecko (parent-child).
2. **Platforma i język niezależne:**
    - DOM może być używany w dowolnym języku programowania, ale najczęściej jest używany z JavaScript w przeglądarkach internetowych.
3. **Dynamiczna manipulacja:**
    - DOM umożliwia odczytywanie, modyfikowanie, dodawanie i usuwanie elementów oraz atrybutów w czasie rzeczywistym.
4. **Dostęp do stylów:**
    - DOM pozwala na modyfikację stylów CSS przypisanych do elementów.

Przykład:

```
<note>
  <to>Alice</to>
  <from>Bob</from>
  <message>Hello!</message>
</note>
```

DOM reprezentuje to jako drzewo:

- note (element główny)
    - to (element) → Alice (tekst)
    - from (element) → Bob (tekst)
    - message (element) → Hello! (tekst)

Programy mogą używać DOM do manipulacji dokumentem XML w czasie rzeczywistym.

1. Select invalid XML elements and give reasons why it is not valid.
2. Find all syntactical errors, and write in the corrections.
3. How will element <email> show with this CSS?
4. What we can use for add some string after content of element <email>?
5. Create XPath expressions for this XML structure.
6. Create an example of one XML document valid with this DTD.
7. Zasady tworzenia dokumentów XML, projektowanie arkuszy transformacji XSLT, definiowanie struktury dokumentu za pomocą DTD, walidacja dokumentów XML.

1. Select invalid XML elements and give reasons why it is not valid.

Nieprawidłowe elementy XML mogą wynikać z kilku typowych błędów w składni, takich jak:

- **Niezgodność z zasadami hierarchii XML** – elementy muszą być odpowiednio zagnieżdżone i zamknięte.
    - Przykład błędu:

      <name>John</name><age>30

      Powód: Element <age> nie jest poprawnie zamknięty.

- **Brak zamknięcia tagu** – każdy element musi mieć swój tag otwierający i zamykający.
    - Przykład błędu:

      ```
      <book>
        <title>XML Tutorial
      </book>
      ```

      Powód: Element <title> nie jest zamknięty.

- **Nieprawidłowe użycie atrybutów** – atrybuty muszą być zapisane w cudzysłowach.
    - Przykład błędu:

      ```
      <book title=XML>
      ```

      Powód: Atrybut title nie jest zapisany w cudzysłowie.

- **Niewłaściwe znaki w nazwach elementów i atrybutów** – nazwy elementów i atrybutów muszą zaczynać się od litery lub podkreślenia i mogą zawierać tylko litery, cyfry, kreski i podkreślenia.
    - Przykład błędu:

      ```
      <1book>Content</1book>
      ```

      Powód: Nazwa elementu nie może zaczynać się od cyfry.

2. Find all syntactical errors, and write the corrections.

Przykład z błędami składniowymi:

```
<catalog>
  <book>
    <title>XML for Beginners</title>
    <author>John Doe</author>
    <price>29.99
  <book>
</catalog>
```

Błędy:

1. Element <price> nie jest poprawnie zamknięty.
2. Element <book> nie jest zamknięty poprawnie (powinno być </book>).

Poprawiona wersja:

```
<catalog>
  <book>
    <title>XML for Beginners</title>
    <author>John Doe</author>
    <price>29.99</price>
  </book>
</catalog>
```

3. How will element <email> show with this CSS?

Załóżmy, że mamy następujący kod CSS:

```
email {
   color: red;
   font-size: 16px;
   text-decoration: underline;
}
```

Przykładowy element XML:

```
<email>example@example.com</email>
```

Zastosowanie CSS:

- Tekst wewnątrz elementu <email> zostanie wyświetlony w kolorze czerwonym, czcionką o rozmiarze 16 pikseli i podkreślony.

4. What we can use for add some string after content of element <email>?

Aby dodać tekst po zawartości elementu <email>, możemy użyć właściwości CSS ::after, która pozwala na dodanie zawartości po elemencie. Przykład:

```
email::after {
   content: " (verified)";
   color: green;
}
```

Wynik:

- Do elementu <email> zostanie dodany tekst "(verified)" po zawartości, a tekst ten będzie wyświetlony na zielono.

5. Create XPath expressions for this XML structure.

Załóżmy, że mamy następujący XML:

```
<bookstore>
   <book>
      <title>XML for Beginners</title>
      <author>John Doe</author>
      <price>29.99</price>
   </book>
   <book>
      <title>Advanced XML</title>
      <author>Jane Smith</author>
      <price>39.99</price>
   </book>
</bookstore>
```

- **Wyrażenie XPath, które zwróci wszystkie książki (book)**:

/bookstore/book

- **Wyrażenie XPath, które zwróci tytuł pierwszej książki**:

  /bookstore/book[1]/title

- **Wyrażenie XPath, które zwróci autora książki "XML for Beginners"**:

  /bookstore/book[title='XML for Beginners']/author

- **Wyrażenie XPath, które zwróci ceny książek**:

  /bookstore/book/price

6. Create an example of one XML document valid with this DTD.

Załóżmy, że mamy DTD, które definiuje strukturę dokumentu XML:

```
<!ELEMENT bookstore (book+)>
<!ELEMENT book (title, author, price)>
<!ELEMENT title (#PCDATA)>
<!ELEMENT author (#PCDATA)>
<!ELEMENT price (#PCDATA)>
```

Przykład dokumentu XML zgodnego z powyższym DTD:

```
<?xml version="1.0"?>
<!DOCTYPE bookstore [
  <!ELEMENT bookstore (book+)>
  <!ELEMENT book (title, author, price)>
  <!ELEMENT title (#PCDATA)>
  <!ELEMENT author (#PCDATA)>
  <!ELEMENT price (#PCDATA)>
]>
<bookstore>
  <book>
    <title>XML for Beginners</title>
    <author>John Doe</author>
    <price>29.99</price>
  </book>
  <book>
    <title>Advanced XML</title>
    <author>Jane Smith</author>
    <price>39.99</price>
  </book>
</bookstore>
```

7. Zasady tworzenia dokumentów XML, projektowanie arkuszy transformacji XSLT, definiowanie struktury dokumentu za pomocą DTD, walidacja dokumentów XML.

Zasady tworzenia dokumentów XML:

- **Poprawność składniowa**: Dokument XML musi mieć poprawną składnię, z zamkniętymi elementami i prawidłowymi atrybutami.
- **Hierarchia danych**: Struktura dokumentu musi być hierarchiczna, z elementami otwierającymi i zamykającymi.
- **Unikalność nazw**: Nazwy elementów i atrybutów muszą być unikalne w ramach danej struktury.

Projektowanie arkuszy transformacji XSLT:

- **XSLT** pozwala na transformację dokumentów XML do innych formatów, np. HTML.
- **Struktura XSLT**: Arkusz transformacji składa się z szablonów, które odpowiadają za przekształcanie poszczególnych elementów XML na HTML, tekst itp.

Przykład szablonu XSLT:

```
<xsl:template match="/">
  <html>
    <body>
      <h1>Books List</h1>
      <xsl:apply-templates select="bookstore/book"/>
    </body>
  </html>
</xsl:template>
```

Definiowanie struktury dokumentu za pomocą DTD:

- DTD służy do określenia, jakie elementy i atrybuty mogą występować w danym dokumencie XML oraz w jakiej kolejności.
- Może być zdefiniowane wewnętrznie w dokumencie XML lub w osobnym pliku.

Walidacja dokumentów XML:

- Walidacja polega na sprawdzeniu, czy dokument XML spełnia wymagania zawarte w DTD lub XML Schema.
- Dokumenty mogą być walidowane podczas ich tworzenia lub przesyłania za pomocą narzędzi takich jak xmllint lub wbudowanych mechanizmów w przeglądarkach i edytorach XML.

Sieci Komputerowe

1. Model OSI oraz jego warstwy funkcjonalne

Model OSI (Open Systems Interconnection) to teoretyczny model, który opisuje sposób, w jaki systemy komputerowe komunikują się w sieci. Model składa się z 7 warstw, z których każda pełni określoną rolę w procesie komunikacji.

Warstwy OSI:

1. **Warstwa fizyczna (Physical Layer)** - Odpowiada za transmisję sygnałów elektrycznych lub optycznych przez medium fizyczne (np. kable, fale radiowe). Określa, jak dane są fizycznie przesyłane (np. poprzez kable, światłowody).
2. **Warstwa łącza danych (Data Link Layer)** - Zapewnia niezawodną transmisję danych pomiędzy urządzeniami w tej samej sieci. Jest odpowiedzialna za detekcję i korekcję błędów, oraz adresowanie sprzętowe (np. adresy MAC).
3. **Warstwa sieciowa (Network Layer)** - Odpowiada za trasowanie danych, czyli wybór ścieżki przesyłania pakietów w sieci. Używa adresowania logicznego (np. adresy IP).
4. **Warstwa transportowa (Transport Layer)** - Odpowiada za zapewnienie niezawodnej transmisji danych pomiędzy urządzeniami końcowymi. Może zapewniać kontrolę błędów i zarządzanie przepustowością. Protokół TCP (Transmission Control Protocol) działa na tej warstwie.
5. **Warstwa sesji (Session Layer)** - Umożliwia nawiązywanie, zarządzanie i kończenie sesji komunikacyjnych między aplikacjami. Dba o synchronizację i kontrolowanie wymiany danych.
6. **Warstwa prezentacji (Presentation Layer)** - Odpowiada za konwersję danych między różnymi formatami, kompresję, szyfrowanie i deszyfrowanie danych. Jej zadaniem jest zapewnienie, że dane są zrozumiałe dla aplikacji.
7. **Warstwa aplikacji (Application Layer)** - Warstwa, z którą bezpośrednio wchodzi w interakcję użytkownik. Odpowiada za dostęp do usług sieciowych, takich jak poczta e-mail, przeglądanie stron internetowych i przesyłanie plików.

2. Model TCP/IP i różnice w stosunku do OSI

Model TCP/IP (Transmission Control Protocol/Internet Protocol) to zestaw protokołów komunikacyjnych używanych do przesyłania danych w internecie i w innych sieciach komputerowych. Jest to bardziej pragmatyczny model niż OSI.

Warstwy modelu TCP/IP:

1. **Warstwa aplikacji (Application Layer)** - Odpowiada za interakcję aplikacji z użytkownikiem. Obejmuje protokoły takie jak HTTP, FTP, DNS, SMTP.
2. **Warstwa transportowa (Transport Layer)** - Odpowiada za niezawodną komunikację pomiędzy urządzeniami. TCP i UDP działają na tej warstwie.
3. **Warstwa internetowa (Internet Layer)** - Odpowiada za trasowanie pakietów i adresowanie w sieci, używając protokołu IP.
4. **Warstwa łącza danych (Link Layer)** - Odpowiada za przesyłanie danych przez medium fizyczne i zarządzanie dostępem do medium transmisyjnego.

Różnice w stosunku do modelu OSI:

- Model TCP/IP ma tylko 4 warstwy (w porównaniu do 7 w OSI).
- TCP/IP nie rozdziela wyraźnie warstwy sesji i prezentacji, traktując je w ramach warstwy aplikacji.
- Model TCP/IP jest bardziej zbliżony do rzeczywistego wdrożenia protokołów w internecie.

3. Protokół komunikacyjny i jego funkcje w poszczególnych warstwach

Protokół komunikacyjny to zbiór reguł i standardów, które określają sposób wymiany informacji w sieci. W każdej warstwie modelu OSI lub TCP/IP protokoły pełnią różne funkcje:

- **Warstwa fizyczna:** Protokół nie jest ściśle określony, ale obejmuje standardy takie jak Ethernet czy Wi-Fi.
- **Warstwa łącza danych:** Protokół Ethernet, ARP (Address Resolution Protocol) – odpowiadają za przesyłanie danych w sieci lokalnej oraz mapowanie adresów IP na adresy MAC.
- **Warstwa sieciowa:** Protokół IP (Internet Protocol) odpowiada za adresowanie i trasowanie pakietów w sieci. Obejmuje zarówno IPv4, jak i IPv6.
- **Warstwa transportowa:** TCP i UDP – protokoły odpowiadające za kontrolowanie i zapewnianie niezawodności komunikacji, zapewniając segmentację i ponowne przesyłanie danych w przypadku błędów.
- **Warstwa aplikacji:** Protokół HTTP (Hypertext Transfer Protocol), FTP (File Transfer Protocol), SMTP (Simple Mail Transfer Protocol) – protokoły zapewniające komunikację w aplikacjach internetowych, np. przesyłanie plików, wysyłanie poczty e-mail, przeglądanie stron WWW.

4. Adresacja IP: IPv4 i IPv6 (struktura, klasy adresów, CIDR)

- **IPv4 (Internet Protocol version 4):**
  - Składa się z 32-bitowego adresu, który jest zapisywany w postaci czterech oktetów oddzielonych kropkami, np. 192.168.1.1.
  - **Klasy adresów IPv4:**
    - **Klasa A**: Adresy od 1.0.0.0 do 127.255.255.255.
    - **Klasa B**: Adresy od 128.0.0.0 do 191.255.255.255.
    - **Klasa C**: Adresy od 192.0.0.0 do 223.255.255.255.
    - **Klasa D**: Adresy od 224.0.0.0 do 239.255.255.255 (adresy multicast).
    - **Klasa E**: Adresy od 240.0.0.0 do 255.255.255.255 (adresy zarezerwowane).
- **IPv6 (Internet Protocol version 6):**
  - Składa się z 128-bitowego adresu, zapisanego w postaci 8 grup po 4 znaki szesnastkowe, np. 2001:0db8:85a3:0000:0000:8a2e:0370:7334.
  - **CIDR (Classless Inter-Domain Routing)**: Umożliwia elastyczne przypisywanie adresów bez podziału na klasy. Wskazuje liczbę bitów przeznaczoną na adres sieci, np. 192.168.1.0/24.

5. Podstawowe protokoły warstwy transportowej (TCP, UDP) i różnice między nimi

- **TCP (Transmission Control Protocol):**
  - Zapewnia niezawodną komunikację, segmentując dane na mniejsze pakiety.
  - Zawiera mechanizmy kontroli błędów i retransmisji pakietów.
  - Wymaga ustalenia połączenia przed wysłaniem danych (handshake).
- **UDP (User Datagram Protocol):**
  - Jest protokołem bezpołączeniowym, nie gwarantuje niezawodności ani kolejkowania pakietów.
  - Działa szybciej, ponieważ nie ma mechanizmów retransmisji.

- Używany w sytuacjach, gdzie szybkość transmisji jest ważniejsza niż niezawodność (np. streaming, VoIP).

Różnice:

- TCP zapewnia niezawodność, UDP jest szybszy, ale mniej niezawodny.
- TCP jest stosowany w sytuacjach wymagających pewności dostarczenia danych, natomiast UDP w komunikacjach, które mogą tolerować utratę danych (np. streaming audio i wideo).

6. Protokoły aplikacyjne (HTTP, FTP, DNS, SMTP) i ich funkcje w komunikacji

- **HTTP (Hypertext Transfer Protocol):**
  - Protokół służący do przesyłania stron WWW. Działa w modelu klient-serwer.
- **FTP (File Transfer Protocol):**
  - Protokół umożliwiający przesyłanie plików pomiędzy komputerami w sieci.
- **DNS (Domain Name System):**
  - Protokół odpowiedzialny za tłumaczenie nazw domenowych na adresy IP.
- **SMTP (Simple Mail Transfer Protocol):**
  - Protokół używany do wysyłania wiadomości e-mail.

7. Topologie sieci (gwiazda, magistrala, pierścień, siatka) i ich zastosowania

- **Gwiazda (Star):**
  - Wszystkie urządzenia są połączone z centralnym urządzeniem (np. switchem). Łatwo zarządzać, ale awaria urządzenia centralnego powoduje awarię całej sieci.
- **Magistrala (Bus):**
  - Wszystkie urządzenia są połączone do jednej linii transmisyjnej. Tanie i łatwe w implementacji, ale trudne do rozbudowy.
- **Pierścień (Ring):**
  - Urządzenia są połączone w jeden cykliczny łańcuch. Problemy z jednym urządzeniem mogą zatrzymać całą sieć.
- **Siatka (Mesh):**
  - Każde urządzenie jest połączone z każdym innym. Zapewnia dużą niezawodność, ale jest kosztowna.

8. Urządzenia sieciowe (routery, switche, huby, access pointy) i ich funkcje

- **Router**: Urządzenie odpowiedzialne za trasowanie pakietów pomiędzy różnymi sieciami. Łączy różne sieci, takie jak lokalne i internetowe.
- **Switch**: Przełącznik, który łączy urządzenia w sieci lokalnej (LAN) i przesyła dane na podstawie adresu MAC.
- **Hub**: Przełącznik działający na poziomie fizycznym. Przesyła sygnały do wszystkich urządzeń w sieci, co może powodować kolizje.
- **Access Point (AP)**: Urządzenie, które umożliwia bezprzewodowe połączenie z siecią (np. Wi-Fi). Umożliwia urządzeniom komunikowanie się z siecią lokalną bez użycia kabli.

Grupy kodów stanu HTTP:

1xx – Informacyjne

- Informują klienta, że żądanie zostało odebrane i jest przetwarzane.
- Przykłady:
    - **100 Continue**: Serwer otrzymał początkową część żądania i klient może kontynuować wysyłanie danych.
    - **101 Switching Protocols**: Serwer zmienia protokół komunikacji (np. z HTTP na WebSocket).

2xx – Sukces

- Wskazują, że żądanie klienta zostało pomyślnie przetworzone.
- Przykłady:
    - **200 OK**: Żądanie zostało przetworzone pomyślnie (np. pobranie strony lub przesłanie formularza).
    - **201 Created**: Żądanie zakończyło się utworzeniem nowego zasobu (np. dodanie wpisu w bazie danych).
    - **204 No Content**: Żądanie zostało przetworzone, ale serwer nie ma treści do zwrócenia (często używane w żądaniach AJAX).

3xx – Przekierowania

- Informują, że klient musi podjąć dodatkowe kroki (np. podążyć za innym URL-em).
- Przykłady:
    - **301 Moved Permanently**: Zasób został trwale przeniesiony pod inny URL.
    - **302 Found**: Tymczasowe przekierowanie do innego URL.
    - **304 Not Modified**: Zasób nie został zmodyfikowany, klient może użyć wersji z pamięci podręcznej.

4xx – Błędy klienta

- Wskazują na problem z żądaniem klienta.
- Przykłady:
    - **400 Bad Request**: Żądanie jest nieprawidłowe lub niezgodne z wymaganiami serwera.
    - **401 Unauthorized**: Klient musi się uwierzytelnić (np. zalogować).
    - **403 Forbidden**: Dostęp do zasobu jest zabroniony, mimo że klient jest poprawnie uwierzytelniony.
    - **404 Not Found**: Serwer nie może znaleźć żądanego zasobu (np. błędny URL).

5xx – Błędy serwera

- Oznaczają, że serwer nie może przetworzyć poprawnego żądania z powodu problemów wewnętrznych.
- Przykłady:

- **500 Internal Server Error**: Ogólny błąd serwera, zazwyczaj z powodu błędów w kodzie aplikacji.
- **502 Bad Gateway**: Serwer pośredniczący (np. proxy) otrzymał niepoprawną odpowiedź od innego serwera.
- **503 Service Unavailable**: Serwer jest tymczasowo niedostępny (np. z powodu przeciążenia lub konserwacji).
- **504 Gateway Timeout**: Serwer pośredniczący nie otrzymał odpowiedzi od innego serwera w odpowiednim czasie.

6xx – Niestandardowe

- Kody te nie są oficjalnie zdefiniowane w standardzie HTTP i mogą być używane w specyficznych systemach. Ich znaczenie zależy od konkretnej implementacji.

Jak działają kody odpowiedzi HTTP?

1. **Wysłanie żądania przez klienta:**
   - Klient (np. przeglądarka) wysyła żądanie do serwera, używając HTTP (np. GET /index.html).
2. **Przetwarzanie przez serwer:**
   - Serwer interpretuje żądanie i podejmuje odpowiednie działania (np. wysyłanie pliku HTML, zapis danych do bazy).
3. **Odpowiedź serwera:**
   - Serwer zwraca odpowiedź zawierającą kod stanu HTTP oraz opcjonalną treść.

1. Planowanie sieci w oparciu o okablowanie strukturalne.
2. Planowanie systemu adresacji IP dla podsieci komputerowych.
3. Omów różnicę między protokołem TCP a UDP.
4. Omów dwie wybrane topologie sieciowe, wymień wady i zalety każdej z nich.
5. Omów pochodzenie wpisów do tablicy routingu i zasada jej działania.
6. Omów system nazw domenowych (DNS).

1. Planowanie sieci w oparciu o okablowanie strukturalne

Okablowanie strukturalne to ujednolicony system okablowania w sieci komputerowej, który zapewnia elastyczność, porządek i możliwość łatwego rozwoju infrastruktury. Planowanie sieci oparte na okablowaniu strukturalnym polega na zaprojektowaniu architektury, która umożliwia skalowanie, prostą konserwację i elastyczność.

Elementy okablowania strukturalnego:

- **Kable** – używa się różnych typów kabli (np. miedzianych, światłowodowych) w zależności od wymagań sieci.
- **Paneli krosowych (patch panel)** – umożliwiają centralne zarządzanie połączeniami w sieci.
- **Gniazda** – do podłączenia urządzeń końcowych.
- **Szafy rackowe** – do umieszczenia urządzeń sieciowych (np. routerów, switchy) i paneli krosowych.

- **Okablowanie pionowe i poziome** – okablowanie pionowe (backbone) łączy szafy rackowe z różnymi segmentami sieci, a okablowanie poziome łączy urządzenia końcowe z siecią.

Korzyści:

- Ujednolicenie kabli, co ułatwia zarządzanie i modernizację sieci.
- Możliwość łatwego rozbudowywania sieci.
- Zwiększenie wydajności i niezawodności sieci.

2. Planowanie systemu adresacji IP dla podsieci komputerowych

Planowanie adresacji IP dla podsieci jest istotnym elementem projektowania sieci komputerowej, pozwalającym na odpowiednie podzielenie dużych sieci na mniejsze podsieci.

Elementy adresacji IP:

- **Adres IP**: unikalny adres, który identyfikuje urządzenie w sieci.
- **Maska podsieci**: Określa, które bity adresu IP są przeznaczone na adres sieciowy, a które na adres urządzenia.
- **Podsieć**: Wynika z podziału większej sieci na mniejsze fragmenty. Na przykład, sieć 192.168.1.0/24 może zostać podzielona na mniejsze podsieci, takie jak 192.168.1.0/26, 192.168.1.64/26 itd.

Zasady planowania:

1. **Określenie wymagań sieci** – liczba urządzeń, potrzeba izolacji sieci.
2. **Podział sieci** – na podstawie maski podsieci tworzymy odpowiednią liczbę podsieci.
3. **Rezerwowanie adresów** – adresy IP są przypisywane z określonego zakresu, a adresy bram i serwerów powinny być rezerwowane.

Przykład podziału:

- Sieć 192.168.0.0/24 ma 256 adresów, które można podzielić na cztery podsieci /26, każda z 64 adresami (62 dostępne dla urządzeń).

3. Omów różnicę między protokołem TCP a UDP

TCP (Transmission Control Protocol):

- **Połączeniowy**: Wymaga nawiązania połączenia pomiędzy nadawcą a odbiorcą (trójstronny handshake).
- **Niezawodność**: Gwarantuje, że dane dotrą do odbiorcy (poprzez kontrolę błędów i retransmisje).
- **Kontrola przepustowości**: Zapewnia kontrolę nad przepustowością, zapobiegając zatorom w sieci.
- **Użycie**: Stosowany tam, gdzie ważna jest niezawodność (np. HTTP, FTP, SMTP).

UDP (User Datagram Protocol):

- **Bezpołączeniowy**: Nie wymaga ustanawiania połączenia przed transmisją danych.
- **Brak niezawodności**: Nie zapewnia gwarancji dostarczenia danych ani kontroli błędów (brak retransmisji).
- **Brak kontroli przepustowości**: Może prowadzić do przeciążenia sieci, ale jest szybszy.
- **Użycie**: Stosowany tam, gdzie szybkość jest ważniejsza niż niezawodność (np. streaming, VoIP).

Różnice:

- TCP jest bardziej niezawodny, ale wolniejszy, podczas gdy UDP jest szybszy, ale mniej niezawodny.

4. Omów dwie wybrane topologie sieciowe, wymień wady i zalety każdej z nich

1. **Topologia gwiazdy (Star):**
   - **Opis**: Wszystkie urządzenia są połączone z centralnym urządzeniem (np. switchem lub routerem).
   - **Zalety**:
     - Łatwość w zarządzaniu i rozbudowie.
     - Awaria jednego urządzenia nie wpływa na całą sieć.
   - **Wady**:
     - Awaria urządzenia centralnego powoduje całkowite zakłócenie komunikacji w sieci.
     - Większe wymagania dotyczące okablowania.
2. **Topologia magistrali (Bus):**
   - **Opis**: Wszystkie urządzenia są podłączone do jednej linii transmisyjnej.
   - **Zalety**:
     - Niskie koszty okablowania.
     - Łatwość instalacji w małych sieciach.
   - **Wady**:
     - Awaria linii transmisyjnej może spowodować zakłócenia w komunikacji.
     - Problemy z wydajnością w dużych sieciach.

   Siatka
   W topologii siatki każde urządzenie w sieci jest bezpośrednio połączone z każdym innym urządzeniem.
   pierścien
   W topologii pierścienia każde urządzenie w sieci jest połączone z dwoma innymi urządzeniami, tworząc zamknięty cykl (pierścień)

Topologia	Zalety	Wady
Pierścień	- Prosta do implementacji - Równomierne rozłożenie obciążenia - Mniejsze zapotrzebowanie na urządzenia pośredniczące	- Awaria pojedynczego urządzenia lub kabla blokuje całą sieć - Trudności w rozbudowie - Wydajność w dużych sieciach
Siatka	- Wysoka niezawodność - Wielokrotne ścieżki do przesyłania danych - Brak centralnego punktu awarii	- Wysoki koszt - Złożoność w skalowaniu i zarządzaniu - Wysokie wymagania sprzętowe

5. Omów pochodzenie wpisów do tablicy routingu i zasada jej działania

Tablica routingu zawiera informacje o tym, jak routery przekierowują pakiety w sieci. Istnieją różne źródła wpisów do tablicy routingu:

1. **Statyczne wpisy routingu**: Administrator ręcznie ustawia trasy w tablicy routingu. Są one stałe i nie zmieniają się dynamicznie.
2. **Dynamiczne wpisy routingu**: Routery mogą automatycznie dodawać wpisy do tablicy routingu na podstawie protokołów routingu (np. OSPF, BGP, RIP). Wpisy te mogą się zmieniać w zależności od zmieniającej się topologii sieci.

Zasada działania:

- **Router** analizuje nagłówek pakietu i na podstawie adresu docelowego decyduje, jaką trasą go wysłać.
- Router sprawdza wpisy w tablicy routingu, aby znaleźć najbliższą trasę do celu i przekazuje pakiet w odpowiednie miejsce.

6. Omów system nazw domenowych (DNS)

DNS (Domain Name System) to system, który umożliwia zamianę nazw domenowych (np. www.example.com) na adresy IP, które są używane w procesie routingu pakietów w sieci.

Działanie DNS:

1. **Rozwiązywanie nazw**: Kiedy użytkownik wpisuje nazwę domeny w przeglądarkę, system DNS tłumaczy tę nazwę na adres IP, który pozwala na połączenie z odpowiednim serwerem.
2. **Struktura DNS**: DNS jest hierarchiczny:
    - **Root (korzeń)**: Najwyższy poziom w strukturze DNS.
    - **Serwery główne**: Odpowiadają za przypisanie domen najwyższego poziomu (np. .com, .org).
    - **Serwery nazw domen**: Odpowiadają za przypisanie nazw domen do adresów IP.

Typy rekordów DNS:

- **A** (Address): Zawiera adres IP powiązany z domeną.
- **MX** (Mail Exchange): Określa serwery poczty e-mail związane z domeną.
- **CNAME** (Canonical Name): Określa aliasy dla domen.

DNS jest fundamentalnym elementem działania internetu, ponieważ umożliwia łatwe zarządzanie nazwami domenowymi i ich mapowanie na adresy IP, co ułatwia komunikację w sieci.

Języki skryptowe

1. Wbudowane typy, klasy i struktury danych oraz ich różne odmiany. Zakresy zmiennych (funkcja, moduł, domknięcie, itd.)

- **Wbudowane typy danych**: Python oferuje różnorodne wbudowane typy danych:
  - **Numeryczne**: int (liczby całkowite), float (liczby zmiennoprzecinkowe), complex (liczby zespolone).
  - **Tekstowe**: str (łańcuchy znaków).
  - **Logiczne**: bool (prawda/fałsz).
  - **Struktury danych**: list (lista), tuple (krotka), set (zbiór), dict (słownik).
  - **Inne typy**: NoneType (specjalny typ reprezentujący brak wartości).
- **Zakresy zmiennych**: W Pythonie zmienne mogą mieć różne zakresy:
  - **Zakres funkcji**: Zmienna zadeklarowana wewnątrz funkcji jest dostępna tylko w tej funkcji.
  - **Zakres modułu**: Zmienna zadeklarowana w module jest dostępna w obrębie całego tego modułu.
  - **Zakres domknięcia (closure)**: Zmienna zadeklarowana w funkcji zagnieżdżonej, która jest dostępna w funkcji zewnętrznej, która ją zdefiniowała.
  - **Zakres globalny**: Zmienna zadeklarowana na poziomie skryptu lub programu jest dostępna globalnie.
  - **Zakres lokalny**: Zmienna zadeklarowana wewnątrz funkcji (lokalna zmienna) jest dostępna tylko w tej funkcji.

1. Listy

Listy to uporządkowane, zmienne struktury danych, które mogą przechowywać elementy różnych typów.

Tworzenie listy:

```
my_list = [1, 2, 3, 4]  # Lista z elementami
```

Działania na listach:

- **Dodawanie elementów**:
  - append(element) – dodaje element na końcu listy.
  - insert(index, element) – dodaje element na określoną pozycję w liście.
  - extend(iterable) – dodaje wszystkie elementy z iterowalnego obiektu na końcu listy.

```
my_list.append(5)  # Dodaje 5 na koniec
my_list.insert(2, 10)  # Dodaje 10 na pozycję 2
my_list.extend([6, 7])  # Rozszerza listę o elementy [6, 7]
```

- **Usuwanie elementów**:
  - remove(value) – usuwa pierwszy wystąpienie elementu o wartości value.
  - pop(index) – usuwa element na danej pozycji (domyślnie usuwa ostatni element).
  - clear() – usuwa wszystkie elementy z listy.

```
my_list.remove(10)  # Usuwa pierwsze wystąpienie 10
my_list.pop(2)  # Usuwa element na pozycji 2
my_list.clear()  # Usuwa wszystkie elementy
```

- **Dostęp do elementów**:
  - Indeksowanie: my_list[index] – dostęp do elementu na określonym indeksie.
  - Slicing: my_list[start:end] – dostęp do podlisty od start do end (bez end).

```
print(my_list[0])  # Pierwszy element
print(my_list[1:3])  # Elementy od indeksu 1 do 2
```

- **Operacje na listach**:
  - len(my_list) – zwraca długość listy.
  - in – sprawdza, czy element znajduje się w liście.
  - not in – sprawdza, czy element nie znajduje się w liście.

```
print(len(my_list))  # Długość listy
print(3 in my_list)  # Czy 3 jest w liście?
```

- **Sortowanie i odwracanie**:
  - sort() – sortuje listę.
  - reverse() – odwraca listę.

```
my_list.sort()  # Sortowanie rosnąco
my_list.reverse()  # Odwrócenie kolejności elementów
```

2. Krotki

Krotki są niezmiennymi, uporządkowanymi kolekcjami. Po utworzeniu krotki nie można zmieniać jej elementów.

Tworzenie krotki:

```
my_tuple = (1, 2, 3)
```

Działania na krotkach:

- **Dostęp do elementów**:
  - Indeksowanie: my_tuple[index] – dostęp do elementu na określonym indeksie.
  - Slicing: my_tuple[start:end] – dostęp do podkrotki od start do end (bez end).

```
print(my_tuple[0])  # Pierwszy element
print(my_tuple[1:3])  # Podkrotka od indeksu 1 do 2
```

- **Operacje na krotkach**:
  - len(my_tuple) – zwraca długość krotki.
  - in – sprawdza, czy element znajduje się w krotce.
  - not in – sprawdza, czy element nie znajduje się w krotce.

```
print(len(my_tuple))  # Długość krotki
print(3 in my_tuple)  # Czy 3 jest w krotce?
```

- **Łączenie krotek**:
    - Krotki można łączyć za pomocą operatora +.
    - Powielać krotki za pomocą operatora *.

```
my_tuple2 = (4, 5)
new_tuple = my_tuple + my_tuple2  # Łączenie krotek
print(new_tuple)  # Wynik: (1, 2, 3, 4, 5)

repeated_tuple = my_tuple * 2  # Powtarzanie krotki
print(repeated_tuple)  # Wynik: (1, 2, 3, 1, 2, 3)
```

3. Zbiory

Zbiory to nieuporządkowane kolekcje unikalnych elementów. Nie zachowują one kolejności elementów.

Tworzenie zbioru:

```
my_set = {1, 2, 3}
```

Działania na zbiorach:

- **Dodawanie elementów**:
    - add(element) – dodaje element do zbioru.
    - update(iterable) – dodaje wszystkie elementy z iterowalnego obiektu do zbioru.

```
my_set.add(4)
my_set.update([5, 6])
```

- **Usuwanie elementów**:
    - remove(value) – usuwa element, ale rzuca wyjątek, jeśli element nie istnieje.
    - discard(value) – usuwa element, ale nie rzuca wyjątku.
    - pop() – usuwa losowy element.
    - clear() – usuwa wszystkie elementy ze zbioru.

```
my_set.remove(4)
my_set.discard(6)
my_set.pop()
my_set.clear()
```

- **Operacje matematyczne na zbiorach**:
    - **Unia** (union(), |) – połączenie dwóch zbiorów.
    - **Przecięcie** (intersection(), &) – elementy wspólne dla obu zbiorów.
    - **Różnica** (difference(), -) – elementy tylko w pierwszym zbiorze.
    - **Różnica symetryczna** (symmetric_difference(), ^) – elementy tylko w jednym zbiorze.
    - **Podzbiór** (issubset()) – sprawdzenie, czy pierwszy zbiór jest podzbiorem drugiego.
    - **Superset** (issuperset()) – sprawdzenie, czy pierwszy zbiór jest supersetem drugiego.
    - **Brak wspólnych elementów** (isdisjoint()) – sprawdzenie, czy zbiory nie mają wspólnych elementów.

```
set1 = {1, 2, 3}
set2 = {3, 4, 5}
```

```
print(set1 | set2)  # Unia
print(set1 & set2)  # Przecięcie
print(set1 - set2)  # Różnica
print(set1 ^ set2)  # Różnica symetryczna
```

4. Słowniki

Słowniki to kolekcje, które przechowują pary klucz-wartość, umożliwiając szybki dostęp do wartości na podstawie klucza.

Tworzenie słownika:

```
my_dict = {'a': 1, 'b': 2, 'c': 3}
```

Działania na słownikach:

- **Dodawanie elementów**:
    - my_dict[key] = value – dodanie nowej pary klucz-wartość.

```
my_dict['d'] = 4
```

- **Usuwanie elementów**:
    - del my_dict[key] – usuwa parę klucz-wartość na podstawie klucza.
    - pop(key) – usuwa parę klucz-wartość i zwraca wartość.
    - popitem() – usuwa i zwraca ostatnią parę klucz-wartość.

```
del my_dict['a']
my_dict.pop('b')
my_dict.popitem()
```

- **Dostęp do elementów**:
    - my_dict[key] – dostęp do wartości na podstawie klucza.
    - get(key) – dostęp do wartości, zwraca None, jeśli klucz nie istnieje.
    - keys() – zwraca wszystkie klucze.
    - values() – zwraca wszystkie wartości.
    - items() – zwraca wszystkie pary klucz-wartość.

```
print(my_dict['c'])  # Dostęp do wartości
print(my_dict.get('d', 'Brak'))  # Pobranie wartości z domyślnym tekstem

print(my_dict.keys())   # Zwraca klucze
print(my_dict.values()) # Zwraca wartości
print(my_dict.items())  # Zwraca pary klucz-wartość
```

- **Sprawdzanie obecności klucza**:
    - in – sprawdza, czy klucz znajduje się w słowniku.

```
print('c' in my_dict)  # Wynik: True
```

2. Sterowanie przebiegiem programu i różne aspekty wykorzystania pętli

- **Pętle**:
  - **for**: Używana do iteracji po elementach kolekcji (listy, słowniki, krotki, ciągi znaków) lub w zadanej liczbie.

    ```
    for i in range(5):
      print(i)
    ```

  - **while**: Używana, gdy nie znamy z góry liczby iteracji i potrzebujemy powtarzać blok kodu dopóki warunek jest spełniony.

    ```
    count = 0
    while count < 5:
      print(count)
      count += 1
    ```

- **break i continue**:
  - **break**: Przerywa działanie pętli natychmiastowo.
  - **continue**: Pomija bieżącą iterację pętli i przechodzi do następnej.
- **Pętla w połączeniu z funkcjami i zakresami**: Często pętle są używane w połączeniu z funkcjami, umożliwiając dynamiczne operacje na danych. Na przykład, można wykorzystywać pętle do generowania wyników w zależności od zmieniającego się stanu zmiennych.

3. Funkcje i podstawowy mechanizm obsługi parametrów, słowa kluczowe. Funkcje rekurencyjne w języku Python

- **Funkcje**: W Pythonie funkcje definiuje się za pomocą słowa kluczowego def:

  ```
  def greet(name):
    return f"Hello, {name}!"
  ```

- **Parametry funkcji**:
  - **Pozycjonowane**: Parametry przekazywane są według ich kolejności.
  - **Domyślne**: Parametry mogą mieć wartości domyślne, które są używane, gdy argument nie jest podany.

    ```
    def greet(name="Guest"):
      return f"Hello, {name}!"
    ```

- **Słowa kluczowe**: Parametry mogą być przekazywane za pomocą nazwanych argumentów.

  ```
  def greet(name, age):
    return f"{name} is {age} years old"
  ```

- **Funkcje rekurencyjne**: Funkcje rekurencyjne wywołują same siebie, aż do osiągnięcia warunku zakończenia. Przykład funkcji obliczającej silnię:

```
def factorial(n):
    if n == 0:
        return 1
    else:
        return n * factorial(n - 1)
```

4. Wyrażenia listowe i generatorowe. Filtrowanie i transformacja danych

- **Wyrażenia listowe** (list comprehensions): Skrócona forma pętli do tworzenia listy.

```
squares = [x ** 2 for x in range(10)]  # Lista kwadratów liczb od 0 do 9
```

- **Wyrażenia generatorowe** (generator expressions): Podobne do wyrażeń listowych, ale tworzące obiekt generatora, który nie przechowuje danych w pamięci.

```
squares = (x ** 2 for x in range(10))  # Generator kwadratów liczb
```

- **Filtrowanie**: Używanie if w wyrażeniach listowych i generatorowych do selekcji danych spełniających określony warunek.

```
even_squares = [x ** 2 for x in range(10) if x % 2 == 0]  # Tylko parzyste liczby
```

- **Transformacja**: Można używać funkcji do transformacji danych w wyrażeniach listowych lub generatorowych. Na przykład, zmiana wszystkich elementów listy na ich podwojoną wartość:

```
doubled = [x * 2 for x in range(10)]
```

5. Wykorzystanie funkcji anonimowych (np. w sortowaniu danych)

- **Funkcje anonimowe**: W Pythonie funkcje anonimowe tworzy się za pomocą lambda. Są to funkcje bez nazwy, które wykonują jedno wyrażenie.

```
add = lambda x, y: x + y
print(add(2, 3))  # Wynik: 5
```

- **Zastosowanie w sortowaniu**: Funkcje anonimowe są używane w sortowaniu z niestandardowym kryterium. Na przykład, sortowanie listy krotek według drugiego elementu:

```
data = [(1, 2), (4, 1), (5, 0)]
data.sort(key=lambda x: x[1])  # Sortowanie według drugiego elementu
```

6. Obsługa plików. Serializacja obiektów. Tworzenie modułów i pakietów

- **Obsługa plików**: pliki można otwierać, odczytywać, pisać i zamykać za pomocą wbudowanych funkcji.

```
with open('file.txt', 'r') as f:
    content = f.read()
```

- **Serializacja obiektów**: Python wspiera serializację obiektów za pomocą modułu pickle lub json, pozwalając na zapis obiektów do plików i późniejsze ich odczytywanie.

```
import pickle
with open('data.pkl', 'wb') as f:
    pickle.dump(my_object, f)
```

- **Moduły i pakiety**: Moduł to plik z kodem Python (np. mymodule.py), który można zaimportować do innych skryptów. Pakiet to zbiór modułów w katalogu.

```
import mymodule
from mymodule import myfunction
```

7. Dekoratory funkcji i domknięcia, funkcje jako obiekty pierwszej klasy

- **Dekoratory**: Funkcje, które przyjmują inną funkcję jako argument i zwracają funkcję, często używane do modyfikowania lub rozszerzania funkcji bez zmiany jej kodu.

```
def decorator(func):
    def wrapper():
        print("Before function call")
        func()
        print("After function call")
    return wrapper

@decorator
def greet():
    print("Hello!")
```

- **Domknięcia (closures)**: Funkcje, które przechowują odniesienia do zmiennych z zewnętrznych funkcji, które były dostępne w momencie ich tworzenia.

```
def outer():
    x = 10
    def inner():
        return x
    return inner
```

8. Podstawowa obsługa wyjątków (konstrukcja try-except)

- **try-except**: Używa się go do obsługi wyjątków, czyli błędów, które mogą wystąpić podczas wykonywania programu.

```
try:
    result = 10 / 0
except ZeroDivisionError:
    print("Nie można dzielić przez zero")
```

- **else i finally**: W bloku try-except można również używać else (który wykona się, jeśli nie wystąpi wyjątek) oraz finally (który zawsze wykona się, niezależnie od tego, czy wystąpił wyjątek).

```
try:
   # kod
except SomeException:
   # obsługa wyjątku
else:
   # wykona się, jeśli nie wystąpi wyjątek
finally:
   # kod wykona się zawsze
```

9. Dopasowanie wzorców z użyciem match/case

- **match/case**: Nowa konstrukcja wprowadzona w Pythonie 3.10, która pozwala na łatwiejsze dopasowywanie wzorców, podobnie jak w innych językach programowania (np. switch w C).

```
def check_number(value):
   match value:
      case 1:
         return "Jedynka"
      case 2:
         return "Dwójka"
      case _:
         return "Inna liczba"
```

1. Wymień zasadę działania dodawania nowego klucza i wartości do słownika.
2. Opisz zasadę działania interpretera kodu Python.
3. W jaki sposób Python interpretuje funkcję z listą jako domyślny parametr?
4. Omów funkcje i funkcję lambda w języku Python.
5. Omów zastosowanie bloku with w kontekście obsługi plików w języku Python.
6. W jaki sposób można obsłużyć wyjątki w języku Python?
7. Omów podstawowe struktury modyfikowalne i niemodyfikowalne w Pythonie.
8. Omów listy i słowniki składane w Pythonie.

1. Zasada działania dodawania nowego klucza i wartości do słownika

Słowniki (dict) to struktury danych przechowujące pary klucz-wartość. Aby dodać nowy klucz i wartość do słownika, wystarczy przypisać wartość do nowego klucza. Jeśli klucz już istnieje, przypisana wartość zastąpi wartość przypisaną wcześniej. Jeśli klucz nie istnieje, zostanie utworzona nowa para klucz-wartość.

Przykład:

```
my_dict = {"a": 1, "b": 2}
my_dict["c"] = 3  # Dodanie nowego klucza "c" z wartością 3
print(my_dict)  # Wynik: {'a': 1, 'b': 2, 'c': 3}
```

2. Zasada działania interpretera kodu Python

Python jest językiem interpretowanym, co oznacza, że kod źródłowy jest tłumaczony na kod maszynowy podczas jego wykonywania, linia po linii. Główne etapy działania interpretera Python to:

1. **Lexing** – Podział kodu źródłowego na tokeny, które są najmniejszymi jednostkami znaczącymi, takimi jak zmienne, liczby, słowa kluczowe itp.
2. **Parsing** – Tworzenie drzewa składniowego (AST), które przedstawia strukturę programu.
3. **Komplilacja** – Przekładanie drzewa składniowego na bajtkod, który jest zrozumiały dla maszyny wirtualnej Pythona (Python Virtual Machine - PVM).
4. **Wykonywanie** – PVM wykonuje bajtkod na maszynie, interpretując go linia po linii.

Interpreter Pythona działa w trybie REPL (Read, Eval, Print, Loop), co oznacza, że kod jest odczytywany, oceniany, wynik jest drukowany, a następnie pętla jest powtarzana.

3. Jak Python interpretuje funkcję z listą jako domyślny parametr?

W Pythonie, jeśli funkcja ma listę jako domyślny parametr, należy pamiętać, że lista jest obiektem mutowalnym. Oznacza to, że jeśli nie przekażemy argumentu do tej funkcji, lista będzie dzielona między wszystkie wywołania tej funkcji. Zatem, modyfikacje w obrębie jednej instancji funkcji wpływają na listę podczas kolejnych wywołań.

Aby uniknąć tej pułapki, dobrym rozwiązaniem jest użycie `None` jako wartości domyślnej, a następnie tworzenie nowej listy w ciele funkcji.

Przykład złego zachowania:

```
def add_element(element, my_list=[]):
   my_list.append(element)
   return my_list

print(add_element(1))  # Wynik: [1]
print(add_element(2))  # Wynik: [1, 2] – lista jest współdzielona
```

Poprawione rozwiązanie:

```
def add_element(element, my_list=None):
   if my_list is None:
      my_list = []
   my_list.append(element)
   return my_list

print(add_element(1))  # Wynik: [1]
print(add_element(2))  # Wynik: [2] – nowa lista za każdym razem
```

4. Omów funkcje i funkcję lambda w języku Python

- **Funkcje w Pythonie**: Funkcje w Pythonie są definiowane za pomocą słowa kluczowego `def`. Funkcje mogą przyjmować parametry, zwracać wartości i zawierać logikę biznesową.

Przykład funkcji:

```
def greet(name):
    return f"Hello, {name}!"
```

- **Funkcja lambda**: Funkcja lambda jest funkcją anonimową, która może zawierać tylko jedno wyrażenie. Zamiast używać pełnej definicji funkcji, lambda umożliwia szybkie definiowanie funkcji, które są używane w określonych miejscach (np. w wyrażeniach).

Przykład funkcji lambda:

```
square = lambda x: x ** 2
print(square(4))  # Wynik: 16
```

5. Omów zastosowanie bloku with w kontekście obsługi plików w języku Python

Blok with jest używany do zarządzania zasobami, takimi jak pliki, gniazda sieciowe, czy połączenia z bazami danych. Dzięki blokowi with, Python automatycznie zajmuje się otwieraniem i zamykaniem zasobów, co zapewnia poprawność działania programu i zapobiega wyciekom pamięci.

W kontekście plików, with pozwala na bezpieczne otwieranie plików, zapewniając ich zamknięcie nawet w przypadku wystąpienia błędów w trakcie pracy z plikiem.
Przykład:

```
with open('file.txt', 'r') as file:
    content = file.read()
# Po wyjściu z bloku 'with' plik jest automatycznie zamknięty
```

6. W jaki sposób można obsłużyć wyjątki w języku Python?

Obsługa wyjątków w Pythonie odbywa się przy użyciu konstrukcji try, except, else i finally.

- **try**: Blok kodu, w którym może wystąpić wyjątek.
- **except**: Blok, który obsługuje wyjątek, jeśli wystąpi.
- **else**: Blok, który wykonuje się, jeśli nie wystąpił żaden wyjątek.
- **finally**: Blok, który zawsze zostanie wykonany, niezależnie od tego, czy wyjątek wystąpił, czy nie.
  Przykład:

```
try:
    result = 10 / 0
except ZeroDivisionError:
    print("Nie można dzielić przez zero")
else:
    print("Operacja zakończona sukcesem")
finally:
    print("Ten blok zawsze się wykona")
```

7. Omów podstawowe struktury modyfikowalne i niemodyfikowalne w Pythonie

- **Modyfikowalne (mutowalne) typy danych**:
  - **Listy (list)**: Można je modyfikować, dodając, usuwając lub zmieniając elementy.

    - **Słowniki (dict)**: Modyfikowalne struktury danych przechowujące pary klucz-wartość.
    - **Zbiory (set)**: Kolekcje unikalnych elementów, które można modyfikować.
- **Niemodyfikowalne (niemutowalne) typy danych**:
    - **Krotki (tuple)**: Zbiory elementów, które nie mogą być modyfikowane po utworzeniu.
    - **Łańcuchy znaków (str)**: Ciągi znaków są niemodyfikowalne, co oznacza, że nie można ich zmieniać po ich utworzeniu.
    - **Liczby (int, float)**: Liczby są również niemodyfikowalne.

8. Omów listy i słowniki składane w Pythonie

- **Listy składane (list comprehensions)**: Umożliwiają tworzenie list w jednej linii kodu, często z zastosowaniem warunków lub transformacji na elementach.

Przykład:

```
squares = [x**2 for x in range(10)]  # Lista kwadratów liczb od 0 do 9
even_squares = [x**2 for x in range(10) if x % 2 == 0]  # Kwadraty tylko parzystych liczb
```

- **Słowniki składane (dict comprehensions)**: Umożliwiają tworzenie słowników w sposób podobny do list składanych, z kluczami i wartościami.

Przykład:

```
squared_dict = {x: x**2 for x in range(5)}  # Słownik z liczbami i ich kwadratami
print(squared_dict)  # Wynik: {0: 0, 1: 1, 2: 4, 3: 9, 4: 16}
```

Przetwarzanie obrazów cyfrowych

1. Filtracja obrazu (filtry górno- i dolnoprzepustowe)

Filtracja obrazu to proces stosowania filtrów do obrazu w celu modyfikacji jego cech. Filtry te służą do usuwania szumów, wygładzania obrazu lub wydobywania określonych szczegółów, takich jak krawędzie.

Filtry górnoprzepustowe

Filtr górnoprzepustowy przepuszcza wysokoczęstotliwościowe składniki obrazu (takie jak krawędzie, detale i zmiany kontrastu) oraz tłumi składniki niskoczęstotliwościowe, które odpowiadają za tło lub gładkie obszary obrazu (np. szum). Jest to przydatne do wykrywania krawędzi i detali w obrazie.

- **Zasada działania**: Filtr górnoprzepustowy działa, stosując maskę (np. filtr Laplace'a, Sobela), która analizuje lokalne zmiany intensywności pikseli obrazu, wzmacniając te zmiany (np. krawędzie), a wygładzając resztę obrazu.
- **Zastosowania**:
    - Wykrywanie krawędzi w obrazach.
    - Usuwanie szumów o niskiej częstotliwości (np. "rozmycie tła").

- **Przykład**: Filtr Sobela stosowany do detekcji krawędzi obrazu, który oblicza gradient intensywności wzdłuż osi x i y.

Filtry dolnoprzepustowe

Filtr dolnoprzepustowy przepuszcza składniki obrazu o niskiej częstotliwości (np. tło) i tłumi składniki wysokoczęstotliwościowe (np. krawędzie). Jest to użyteczne w eliminacji szumów o wysokiej częstotliwości, wygładzaniu obrazów i usuwaniu drobnych szczegółów.

- **Zasada działania**: Filtr dolnoprzepustowy analizuje średnie wartości pikseli w okolicach każdego piksela, aby wygładzić obraz. Filtry takie jak rozmycie Gaussa są przykładem filtrów dolnoprzepustowych, które wygładzają obraz, usuwając szczegóły wysokiej częstotliwości.
- **Zastosowania**:
    - Wygładzanie obrazu.
    - Redukcja szumów.
    - Rozmycie tła.
- **Przykład**: Filtr Gaussa, który wygładza obraz, stosując maskę opartą na funkcji Gaussa, co sprawia, że piksele w centrum maski mają wyższą wagę niż te na obrzeżach.

2. Binaryzacja obrazu

Binaryzacja obrazu to proces przekształcania obrazu w obraz czarno-biały (dwu wartościowy), gdzie każdy piksel jest reprezentowany przez jedną z dwóch możliwych wartości (np. 0 lub 255, czarny lub biały). Jest to istotny krok w wielu algorytmach przetwarzania obrazu, zwłaszcza gdy chcemy wyodrębnić obiekty w obrazie lub wykonać dalsze operacje na kształtach.

Zasada działania:

- **Progiowanie**: Binaryzacja polega na przypisaniu każdemu pikselowi jednej z dwóch wartości na podstawie jego intensywności. Najczęściej stosuje się próg (tzw. próg binarnej klasyfikacji), który decyduje, czy piksel zostanie zakwalifikowany jako biały, czy czarny. Jeśli wartość intensywności piksela jest większa niż ustalony próg, przypisuje się jej wartość 255 (biały), w przeciwnym razie 0 (czarny).
- **Metoda globalna**: Wartość progu jest stała dla całego obrazu. Jest to najprostsza metoda, ale może nie działać dobrze, jeśli obraz ma nierównomierne oświetlenie.
- **Metoda lokalna**: Próg jest obliczany lokalnie dla każdego piksela, co daje lepsze wyniki w przypadku obrazów z nierównomiernym oświetleniem lub kontrastem.
- **Zastosowania**:
    - Wydobycie krawędzi obiektów.
    - Segmentacja tła i obiektów.
    - Przygotowanie do dalszych operacji (np. wykrywanie obiektów, analiza kształtów).
- **Przykład**:

```
import cv2
img = cv2.imread('image.jpg', 0)  # Załaduj obraz w skali szarości
_, binary_img = cv2.threshold(img, 128, 255, cv2.THRESH_BINARY)
```

3. Segmentacja obrazu

Segmentacja obrazu to proces dzielenia obrazu na różne regiony lub segmenty, które mają podobne cechy (np. intensywność pikseli, tekstura, kolor). Celem segmentacji jest uproszczenie obrazu i ułatwienie analizy.

Zasada działania:

Segmentacja jest kluczowym etapem w analizie obrazu, który polega na identyfikacji regionów o podobnych właściwościach, takich jak:

- **Segmentacja na podstawie krawędzi**: wykrywanie krawędzi w obrazie, które oddzielają różne obszary. Do tego celu używa się operatorów takich jak Sobel, Canny itp.
- **Segmentacja na podstawie regionów**: dzielenie obrazu na regiony, które mają podobną intensywność pikseli. Może to obejmować metody takie jak rozrost regionów (region growing) lub podział na segmenty na podstawie histogramu.
- **Metody oparte na klasteryzacji**: takie jak k-means, które grupują piksele w obrazie na podstawie ich cech (np. intensywność, kolor).

Zastosowania:

- Wykrywanie obiektów na tle.
- Analiza medyczna (np. segmentacja organów na obrazach MRI).
- Rozpoznawanie scen w obrazach.
- Przetwarzanie obrazów satelitarnych.
- **Przykład**:

```
import cv2
import numpy as np

img = cv2.imread('image.jpg')
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
_, binary = cv2.threshold(gray, 127, 255, cv2.THRESH_BINARY)
contours, _ = cv2.findContours(binary, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
cv2.drawContours(img, contours, -1, (0,255,0), 3)
```

4. Metody wyszukiwania różnic w obrazach

Wyszukiwanie różnic w obrazach polega na analizie dwóch obrazów, aby znaleźć zmiany, które zaszły między nimi. Jest to przydatne w wielu zastosowaniach, takich jak porównywanie obrazów, detekcja zmian w czasie lub porównywanie obrazów przed i po przetwarzaniu.

Zasada działania:

- **Porównanie obrazów**: Obrazy mogą być porównywane za pomocą różnych metod, takich jak obliczanie różnic pikseli, analiza histogramów, porównanie krawędzi itp. Różnice mogą być oceniane na poziomie poszczególnych pikseli lub regionów.
- **Algorytmy różnicowania**:
    - **Porównanie różnicy pikseli**: Obliczanie różnicy między odpowiadającymi sobie pikselami dwóch obrazów.
    - **Metoda SSIM (Structural Similarity Index)**: Mierzy podobieństwo między dwoma obrazami w sposób uwzględniający ich strukturę, jasność i kontrast.
    - **Porównanie histogramów**: Obrazy mogą być porównywane pod kątem rozkładu intensywności pikseli w całym obrazie (histogram).

Zastosowania:

- Wykrywanie zmian w obrazach w aplikacjach monitoringu.
- Porównywanie zdjęć przed i po przetwarzaniu.
- Analiza różnic między obrazami o różnych poziomach jakości.
- **Przykład**:

```
import cv2
import numpy as np

img1 = cv2.imread('image1.jpg', 0)  # Załaduj oba obrazy w skali szarości
img2 = cv2.imread('image2.jpg', 0)

diff = cv2.absdiff(img1, img2)  # Oblicz różnicę absolutną między obrazami
_, thresh = cv2.threshold(diff, 25, 255, cv2.THRESH_BINARY)  # Proguj różnicę
```

Filtracja obrazu: Używa filtrów górnoprzepustowych i dolnoprzepustowych do usuwania szumów, wykrywania krawędzi i wygładzania obrazów.
Binaryzacja obrazu: Konwertuje obraz na dwuwartościowy, aby ułatwić dalszą analizę (np. segmentację, wykrywanie krawędzi).
Segmentacja obrazu: Dzielenie obrazu na regiony o podobnych cechach w celu analizy, wykrywania obiektów, rozpoznawania kształtów itp.
Metody wyszukiwania różnic: Porównanie dwóch obrazów w celu wykrycia zmian lub różnic między nimi.

Projektownie aplikacji internetowych

1. Zgodność z WCAG 2.1

WCAG (Web Content Accessibility Guidelines) to zestaw wytycznych opracowanych przez W3C (World Wide Web Consortium), które mają na celu zapewnienie dostępności treści internetowych dla osób z różnymi rodzajami niepełnosprawności. WCAG 2.1 jest rozszerzeniem wersji 2.0 i uwzględnia dodatkowe aspekty dostępności, w szczególności związane z urządzeniami mobilnymi oraz użytkownikami z zaburzeniami poznawczymi.

Podstawowe zasady WCAG 2.1:

Wytyczne WCAG 2.1 opierają się na czterech głównych zasadach:

1. **Postrzegalność**:
    - Treść musi być dostrzegalna dla użytkowników, np. za pomocą wzroku, słuchu czy dotyku.
    - **Wymagania**:
        - Teksty alternatywne dla obrazów (np. opis obrazków w atrybucie alt).
        - Napisy do materiałów wideo.
        - Odpowiedni kontrast tekstu i tła (minimalny współczynnik kontrastu wynosi 4.5:1).
        - Responsywne skalowanie treści bez utraty funkcjonalności (do 200%).
2. **Funkcjonalność**:
    - Interfejsy użytkownika i nawigacja muszą być funkcjonalne za pomocą różnych urządzeń wejściowych.
    - **Wymagania**:
        - Strony muszą być w pełni obsługiwalne za pomocą klawiatury (np. za pomocą klawisza Tab).
        - Jasne i logiczne nagłówki oraz kolejność nawigacji.
        - Możliwość pominięcia bloków powtarzających się (np. link "Przejdź do treści").
3. **Zrozumiałość**:
    - Treść i interfejsy muszą być łatwe do zrozumienia.
    - **Wymagania**:
        - Prosty język i zrozumiałe instrukcje.
        - Wyraźne oznaczenia błędów i sugestie ich naprawy.
        - Zapewnienie przewidywalnego działania interfejsu.
4. **Solidność**:
    - Treść musi być wystarczająco solidna, aby była kompatybilna z różnymi technologiami (np. czytnikami ekranowymi).
    - **Wymagania**:
        - Użycie semantycznych elementów HTML (np. <header>, <footer>).
        - Obsługa standardów ARIA (Accessible Rich Internet Applications).

Kryteria sukcesu w WCAG 2.1:

- **Poziom A**: Minimalne wymagania dostępności (np. nawigacja klawiaturą, alternatywne teksty).
- **Poziom AA**: Średni poziom wymagań (np. odpowiedni kontrast, przewidywalność interfejsu).
- **Poziom AAA**: Zaawansowane wymagania dostępności (np. szczegółowe napisy do wideo, brak błędów językowych).

Nowe aspekty w WCAG 2.1:

- Obsługa ekranów dotykowych i urządzeń mobilnych.
- Dostosowanie interfejsu do zaburzeń poznawczych (np. większe przyciski, uproszczona nawigacja).
- Minimalizowanie efektów migotania, co zapobiega wywoływaniu napadów epileptycznych.

Zastosowania:

Zgodność z WCAG 2.1 jest wymagana w wielu krajach w ramach przepisów prawnych (np. ustawa o dostępności cyfrowej w UE). Wdrażanie tych wytycznych zwiększa użyteczność aplikacji i jej dostępność dla wszystkich użytkowników.

2. Wymagania dotyczące ochrony danych osobowych w projektowaniu aplikacji

Ochrona danych osobowych jest kluczowym elementem w projektowaniu aplikacji, szczególnie w kontekście przepisów takich jak RODO (Rozporządzenie Ogólne o Ochronie Danych) w Unii Europejskiej czy CCPA w Kalifornii.

Podstawowe zasady ochrony danych osobowych:

1. **Zasada minimalizacji danych**:
    - Gromadzenie tylko takich danych, które są absolutnie niezbędne do działania aplikacji.
    - Przykład: Aplikacja do logowania powinna wymagać tylko adresu e-mail i hasła, a nie dodatkowych informacji, takich jak wiek czy adres.
2. **Zasada przejrzystości**:
    - Użytkownik musi być w pełni informowany o tym, jakie dane są gromadzone, w jakim celu i kto będzie miał do nich dostęp.
    - Przykład: Wyraźne i zrozumiałe polityki prywatności.
3. **Zasada zgody**:
    - Dane mogą być przetwarzane tylko za zgodą użytkownika, która musi być wyrażona w sposób świadomy i jednoznaczny.
    - Przykład: Checkbox zgody na przetwarzanie danych osobowych, który nie jest domyślnie zaznaczony.
4. **Zasada bezpieczeństwa danych**:
    - Dane osobowe muszą być chronione przed nieautoryzowanym dostępem, wyciekiem lub utratą.
    - Przykład: Szyfrowanie danych w tranzycie (SSL/TLS) i w spoczynku.
5. **Zasada prawa do usunięcia danych (prawo do bycia zapomnianym)**:
    - Użytkownik ma prawo żądać usunięcia swoich danych osobowych z systemu.
    - Przykład: Funkcja w aplikacji umożliwiająca usunięcie konta wraz z danymi.
6. **Zasada przenoszalności danych**:
    - Użytkownik ma prawo otrzymać swoje dane w formacie nadającym się do odczytu maszynowego.

Wymagania techniczne i procesy:

1. **Bezpieczeństwo danych w aplikacji**:
    - **Szyfrowanie danych**:
        - Szyfrowanie danych przechowywanych w bazie (np. hasła hashowane za pomocą algorytmów takich jak bcrypt).
        - Szyfrowanie transmisji danych za pomocą protokołów takich jak HTTPS.
    - **Mechanizmy uwierzytelniania**:
        - Dwuskładnikowe uwierzytelnianie (2FA).
        - Weryfikacja haseł zgodna z najlepszymi praktykami (np. minimalna długość hasła, brak haseł domyślnych).
2. **Anonimizacja i pseudonimizacja danych**:
    - Anonimizacja: Trwałe usunięcie powiązania danych z osobą, której dotyczą.

    - Pseudonimizacja: Ukrycie danych osobowych za pomocą identyfikatorów, które mogą być odwracalne.
3. **Zarządzanie zgodami**:
    - Rejestracja zgód użytkownika na przetwarzanie danych.
    - Przechowywanie historii zgód w sposób audytowalny.
4. **Audyt i monitoring**:
    - Regularne testy bezpieczeństwa aplikacji (np. testy penetracyjne).
    - Śledzenie dostępu do danych i tworzenie dzienników logów.

Przykład wdrożenia ochrony danych w aplikacji:

- **Formularz rejestracji użytkownika**:
    - Pole e-mail i hasło.
    - Checkbox do zgody na przetwarzanie danych z opisem polityki prywatności.
    - Mechanizm CAPTCHA zabezpieczający przed automatycznym rejestrowaniem.
- **Szyfrowanie**:
    - Hasło użytkownika przechowywane w bazie danych jako hash bcrypt.
    - Dane przesyłane za pomocą HTTPS.
- **Prawo do usunięcia danych**:
    - Opcja "Usuń konto" w ustawieniach użytkownika, która kasuje wszystkie dane powiązane z użytkownikiem.

Fizyczne podstawy działania urządzeń informatycznych

1. Zasada działania tranzystora w elektronice cyfrowej

Tranzystor to podstawowy element elektroniczny wykorzystywany w układach cyfrowych, który działa jako przełącznik lub wzmacniacz. Jest to trójkońcówkowy półprzewodnikowy element (najczęściej typu MOSFET lub bipolarnego tranzystora BJT). W elektronice cyfrowej tranzystory MOSFET są najbardziej popularne.

Zasada działania MOSFET w roli przełącznika:

1. **Budowa tranzystora MOSFET**:
    - Składa się z trzech warstw: bramki (gate), drenu (drain) i źródła (source).
    - Bramkę oddziela od kanału warstwa tlenku krzemu, która działa jako izolator.
2. **Praca jako przełącznik**:
    - Przepływ prądu między drenem a źródłem jest kontrolowany przez napięcie przyłożone do bramki.
    - Gdy napięcie na bramce przekracza pewien próg (tzw. napięcie progowe), powstaje kanał przewodzący i tranzystor "włącza się", przewodząc prąd.
    - Gdy napięcie na bramce jest mniejsze od progu, tranzystor pozostaje "wyłączony" i blokuje przepływ prądu.
3. **Zastosowanie w układach cyfrowych**:
    - Tranzystory działają w trybie binarnym: "0" (wyłączony) i "1" (włączony).
    - Używane są do budowy bramek logicznych, rejestrów i innych elementów cyfrowych.

2. Wykorzystanie laserów w odczycie i zapisie danych

Lasery są szeroko stosowane w technologii pamięci optycznych, takich jak płyty CD, DVD i Blu-ray. Ich rola polega na odczycie i zapisie danych za pomocą precyzyjnie ukierunkowanej wiązki światła.

Odczyt danych za pomocą lasera:

1. Płyta optyczna zawiera dane zapisane w postaci mikroskopijnych wgłębień (pit) i płaskich obszarów (land).
2. Wiązka lasera o niskiej mocy jest kierowana na powierzchnię płyty.
3. Odbite światło jest analizowane przez detektor:
    - Wgłębienia odbijają światło w sposób rozproszony, co odpowiada "0".
    - Płaskie obszary odbijają światło w sposób skoncentrowany, co odpowiada "1".

Zapis danych za pomocą lasera:

1. W zapisie wykorzystywany jest laser o wyższej mocy.
2. Wiązka lasera podgrzewa określone miejsca na warstwie materiału, tworząc wgłębienia (pity).
3. Po ostygnięciu dane zostają utrwalone w materiale płyty.

Zalety stosowania laserów:

- Wysoka precyzja odczytu i zapisu.
- Możliwość gęstego zapisu danych.
- Niezawodność dzięki braku mechanicznego kontaktu między głowicą a nośnikiem.

3. Magnetyczne nośniki danych i ich mechanizmy zapisu

Magnetyczne nośniki danych, takie jak dyski twarde (HDD) czy taśmy magnetyczne, wykorzystują właściwości magnetyzmu do przechowywania informacji binarnych.

Mechanizm zapisu danych:

1. Nośnik zawiera warstwę magnetyczną podzieloną na mikroskopijne obszary zwane domenami magnetycznymi.
2. Głowica zapisu generuje pole magnetyczne, które zmienia kierunek namagnesowania domen.
3. Kierunek namagnesowania odpowiada wartościom binarnym:
    - Jeden kierunek = "1".
    - Drugi kierunek = "0".

Mechanizm odczytu danych:

1. Głowica odczytu mierzy zmiany w kierunku namagnesowania domen podczas przesuwania się nośnika.
2. Wygenerowane sygnały elektryczne są interpretowane jako dane binarne.

Zalety magnetycznych nośników danych:

- Duża pojemność.
- Relatywnie niski koszt przechowywania danych.
- Długa trwałość danych przy odpowiednich warunkach przechowywania.

4. Fale elektromagnetyczne w światłowodach i ich rola w transmisji danych

Światłowody to cienkie włókna szklane lub plastikowe, które wykorzystują fale elektromagnetyczne w paśmie światła do przesyłania danych na duże odległości z minimalnymi stratami.

Zasada działania światłowodu:

1. Fale elektromagnetyczne (światło) są wprowadzane do rdzenia światłowodu.
2. Dzięki zjawisku całkowitego wewnętrznego odbicia światło odbija się od ścianek rdzenia i przemieszcza się wzdłuż włókna.
3. Na drugim końcu światłowodu odbierane światło jest przekształcane na sygnały elektryczne.

Zalety transmisji danych w światłowodach:

- Bardzo duża przepustowość (możliwość przesyłania danych z prędkościami sięgającymi terabitów na sekundę).
- Niska tłumienność (małe straty sygnału na odległość).
- Brak zakłóceń elektromagnetycznych.
- Wysoka odporność na podsłuchy dzięki trudności w przechwyceniu sygnału optycznego.

5. Ekrany LCD i LED – zasady działania oraz wykorzystanie efektów optycznych w technologii wyświetlaczy

Ekrany LCD i LED są powszechnie stosowane w urządzeniach elektronicznych, takich jak telewizory, monitory czy smartfony. Oba rodzaje ekranów wykorzystują różne technologie oparte na efektach optycznych.

Ekrany LCD (Liquid Crystal Display):

1. **Zasada działania**:
    - Składają się z warstw ciekłych kryształów, które modulują światło przechodzące przez ekran.
    - Każdy piksel ekranu jest kontrolowany przez tranzystory TFT (Thin-Film Transistor).
    - Światło emitowane przez podświetlenie (np. świetlówki CCFL lub diody LED) przechodzi przez filtry kolorów (RGB).
2. **Właściwości**:
    - Moduluje polaryzację światła przy użyciu elektrycznego pola, co pozwala na kontrolę ilości światła przechodzącego przez piksel.
3. **Zalety**:
    - Niskie zużycie energii.
    - Cienka konstrukcja.
    - Dobre odwzorowanie kolorów w jasnych warunkach.
4. **Wady**:

- Ograniczone kąty widzenia.
- Niższy kontrast w porównaniu z technologiami OLED.

Ekrany LED (Light Emitting Diode):

1. **Zasada działania**:
   - Wykorzystują diody LED jako źródło światła, które może być bezpośrednie (w przypadku wyświetlaczy OLED) lub służyć jako podświetlenie (dla wyświetlaczy LCD LED).
   - W OLED każda dioda emituje własne światło, eliminując potrzebę podświetlenia.
2. **Właściwości**:
   - Bardzo wysoki kontrast dzięki możliwości wyłączania pojedynczych pikseli (czyste czernie).
   - Lepsze kąty widzenia w porównaniu do LCD.
3. **Zalety**:
   - Szybszy czas reakcji.
   - Energooszczędność przy wyświetlaniu ciemnych treści.
   - Lekkie i elastyczne konstrukcje.
4. **Wady**:
   - Wyższy koszt produkcji.
   - Potencjalne problemy z trwałością (np. wypalanie pikseli w OLED).

1. Opisz działanie ekranu LCD.
2. Omów konstrukcję oraz zasadę działania dysku twardego HDD.
3. Omów zasadę działania oraz wykorzystanie transformatora.
4. Opisz działanie monitorów QLED.
5. Omów działanie Lasera oraz jego zastosowanie w odtwarzaczach CD.
6. Opisz zasadę działania diody półprzewodnikowej oraz jej zastosowanie do konstrukcji bramek OR i AND.

1. Działanie ekranu LCD

Ekran LCD (Liquid Crystal Display) wykorzystuje ciekłe kryształy do modulowania światła w celu wyświetlania obrazu.

Zasada działania:

1. **Źródło podświetlenia:**
   - Pod ekranem znajduje się podświetlenie, zwykle w postaci diod LED lub świetlówek CCFL.
   - Światło to jest równomiernie rozpraszane przez specjalną warstwę dyfuzyjną.
2. **Warstwa ciekłych kryształów:**
   - Ciekłe kryształy zmieniają swoje właściwości optyczne pod wpływem pola elektrycznego.
   - Każdy piksel składa się z subpikseli odpowiadających kolorom RGB (czerwony, zielony, niebieski).
3. **Polaryzacja światła:**
   - Światło przechodzi przez pierwszą warstwę polaryzującą.

    - Ciekłe kryształy zmieniają kierunek polaryzacji światła w zależności od przyłożonego napięcia.
    - Następnie światło trafia na drugą warstwę polaryzującą, która blokuje lub przepuszcza światło w zależności od polaryzacji.
4. **Filtry kolorów:**
    - Światło przepuszczone przez ciekłe kryształy trafia na filtry kolorów (RGB), które nadają obrazowi odpowiedni kolor.

Zalety:

- Niskie zużycie energii.
- Cienka konstrukcja.
- Wysoka rozdzielczość obrazu.

Wady:

- Ograniczony kontrast.
- Ograniczone kąty widzenia.

2. Konstrukcja oraz zasada działania dysku twardego HDD

Konstrukcja dysku twardego:

- **Talerze magnetyczne:**
    - Obracające się okrągłe dyski pokryte materiałem magnetycznym.
- **Głowice odczytu/zapisu:**
    - Przesuwają się nad powierzchnią talerzy, wykonując operacje odczytu i zapisu.
- **Silnik:**
    - Napędza obrót talerzy z prędkością od 5400 do 15 000 obr./min.
- **Obudowa:**
    - Chroni dysk przed kurzem i uszkodzeniami mechanicznymi.

Zasada działania:

1. **Zapis danych:**
    - Głowica zapisująca generuje pole magnetyczne, które zmienia kierunek namagnesowania domen na talerzu.
    - Kierunki te odpowiadają wartościom binarnym (0 lub 1).
2. **Odczyt danych:**
    - Głowica odczytująca mierzy zmiany pola magnetycznego, przekształcając je na sygnały elektryczne.
    - Sygnały te są interpretowane jako dane binarne.

Zalety HDD:

- Duża pojemność.
- Niski koszt w przeliczeniu na gigabajt.

Wady HDD:

- Wolniejszy odczyt/zapis w porównaniu z SSD.
- Wrażliwość na wstrząsy i uszkodzenia mechaniczne.

3. Zasada działania oraz wykorzystanie transformatora

Zasada działania:

1. **Podstawy:**
    - Transformator działa na zasadzie indukcji elektromagnetycznej.
    - Składa się z dwóch uzwojeń (pierwotnego i wtórnego) nawiniętych na rdzeniu magnetycznym.
2. **Działanie:**
    - Prąd przemienny w uzwojeniu pierwotnym wytwarza zmienne pole magnetyczne w rdzeniu.
    - Pole magnetyczne indukuje napięcie w uzwojeniu wtórnym.
    - Stosunek napięć wyjściowego do wejściowego zależy od liczby zwojów w uzwojeniach.

Wykorzystanie:

- Zmiana napięcia w sieciach energetycznych.
- Zasilanie urządzeń elektronicznych.
- Separacja obwodów elektrycznych w celu ochrony.

4. Działanie monitorów QLED

QLED (Quantum Dot LED) to technologia wyświetlania wykorzystująca kropki kwantowe w połączeniu z podświetleniem LED.

Zasada działania:

1. **Podświetlenie LED:**
    - Źródłem światła jest podświetlenie LED.
2. **Warstwa kropek kwantowych:**
    - Kropki kwantowe to nanokryształy emitujące światło w określonych długościach fali (kolorach) po ich pobudzeniu.
    - Emitują czystsze kolory niż tradycyjne filtry RGB.
3. **Warstwy polaryzujące i ciekłokrystaliczne:**
    - Podobne do tradycyjnych ekranów LCD, gdzie ciekłe kryształy modulują światło.

Zalety QLED:

- Wyższa jasność.
- Bogatsze kolory w porównaniu do tradycyjnego LCD.
- Dłuższa żywotność niż OLED.

Wady QLED:

- Niższy kontrast w porównaniu do OLED.
- Uzależnienie od podświetlenia LED.

5. Działanie lasera oraz jego zastosowanie w odtwarzaczach CD

Zasada działania lasera:

1. **Generowanie światła:**
    - Laser emituje spójne i jednobarwne światło dzięki wzmocnieniu optycznemu w ośrodku aktywnym (np. półprzewodniku lub gazie).
2. **Skupienie wiązki:**
    - Dzięki soczewkom wiązka jest skupiona na bardzo małej powierzchni.

Zastosowanie w odtwarzaczach CD:

1. **Odczyt danych:**
    - Laser jest kierowany na powierzchnię płyty, która zawiera wgłębienia (pity) i płaskie obszary (landy).
    - Odbite światło trafia na detektor, który interpretuje sygnały jako dane binarne.
2. **Zalety:**
    - Bezstykowy odczyt, co minimalizuje zużycie mechaniczne.
    - Wysoka precyzja.

6. Zasada działania diody półprzewodnikowej oraz jej zastosowanie w konstrukcji bramek OR i AND

Zasada działania diody:

1. **Budowa:**
    - Dioda składa się z dwóch warstw półprzewodnikowych: typu P (z nadmiarem dziur) i typu N (z nadmiarem elektronów).
2. **Działanie:**
    - Gdy napięcie przyłożone jest w kierunku przewodzenia (anoda do P, katoda do N), elektrony i dziury przemieszczają się, umożliwiając przepływ prądu.
    - W kierunku zaporowym prąd nie płynie.

Zastosowanie w bramkach logicznych:

1. **Bramka OR:**
    - Dwie diody są podłączone równolegle, każda do jednego wejścia.
    - Jeżeli przynajmniej jedno wejście ma stan wysoki (napięcie), wyjście również jest wysokie.
    - Zasada działania: A+B.
2. **Bramka AND:**
    - Diody są podłączone szeregowo.
    - Prąd przepływa tylko wtedy, gdy oba wejścia mają stan wysoki.
    - Zasada działania: A·B.

Wstęp do programowania w języku Java

1. Podstawowe typy danych oraz instrukcje warunkowe

Podstawowe typy danych:

Java oferuje dwa rodzaje typów danych:

1. **Typy prymitywne:**
   - **Liczbowe całkowite:**
     - byte (8 bitów), short (16 bitów), int (32 bity), long (64 bity).
   - **Liczbowe zmiennoprzecinkowe:**
     - float (32 bity), double (64 bity).
   - **Logiczny:**
     - boolean (wartości true lub false).
   - **Znakowy:**
     - char (16 bitów, reprezentuje jeden znak w kodowaniu Unicode).
2. **Typy referencyjne:**
   - Klasy, interfejsy, tablice i obiekty, np. String.

Instrukcje warunkowe:

1. **if-else:**

```
if (warunek) {
   // Blok kodu, gdy warunek jest spełniony
} else {
   // Blok kodu, gdy warunek nie jest spełniony
}
```

2. **switch:** Używane do wyboru jednego z wielu bloków kodu na podstawie wartości zmiennej.

```
switch (zmienna) {
   case wartość1:
      // Kod dla wartość1
      break;
   case wartość2:
      // Kod dla wartość2
      break;
   default:
      // Kod, gdy żaden warunek nie jest spełniony
}
```

2. Działanie różnych typów pętli w Java

1. **for:** Pętla iteracyjna o znanej liczbie iteracji.

```
for (int i = 0; i < 10; i++) {
   System.out.println(i);
}
```

2. **while:** Pętla, która działa, dopóki warunek jest spełniony.

   ```
   int i = 0;
   while (i < 10) {
      System.out.println(i);
      i++;
   }
   ```

3. **do-while:** Wykonuje blok kodu przynajmniej raz, a następnie sprawdza warunek.

   ```
   int i = 0;
   do {
      System.out.println(i);
      i++;
   } while (i < 10);
   ```

4. **for-each:** Używana do iteracji po kolekcjach lub tablicach.

   ```
   for (String element : lista) {
      System.out.println(element);
   }
   ```

3. Zagadnienie tworzenia klas, obiektów oraz korzystanie z konstruktorów

Tworzenie klasy:

```
public class Klasa {
   int pole;

   // Konstruktor
   public Klasa(int pole) {
      this.pole = pole;
   }

   // Metoda
   public void metoda() {
      System.out.println("Pole: " + pole);
   }
}
```

Tworzenie obiektu:

```
Klasa obiekt = new Klasa(5);
obiekt.metoda();  // Wywołanie metody
```

Konstruktory:

- Specjalne metody używane do inicjalizacji obiektu.

- Domyślny konstruktor (bezargumentowy) jest tworzony automatycznie, jeśli nie zadeklarujemy żadnego innego.

4. Mechanizm dziedziczenia i kompozycji

Dziedziczenie:

Pozwala jednej klasie przejąć pola i metody innej klasy.

```
class KlasaBazowa {
   public void metodaBazowa() {
      System.out.println("Metoda bazowa");
   }
}

class KlasaPochodna extends KlasaBazowa {
   public void metodaPochodna() {
      System.out.println("Metoda pochodna");
   }
}
```

Kompozycja:

Zamiast dziedziczyć, klasa używa innej klasy jako swojego pola.

```
class Silnik {
   public void uruchom() {
      System.out.println("Silnik uruchomiony");
   }
}

class Samochod {
   private Silnik silnik = new Silnik();

   public void jedz() {
      silnik.uruchom();
      System.out.println("Samochód jedzie");
   }
}
```

5. Interfejsy, wyrażenia lambda i klasy wewnętrzne

Interfejsy:

Definiują zestaw metod, które muszą zostać zaimplementowane przez klasy.

```
interface Zwierze {
   void dzwiek();
```

```
}

class Pies implements Zwierze {
   public void dzwiek() {
      System.out.println("Hau hau");
   }
}
```

Wyrażenia lambda:

Skrócona forma wyrażenia dla interfejsów funkcyjnych (z jedną metodą abstrakcyjną).

```
Runnable r = () -> System.out.println("Wykonuję zadanie w wątku");
```

Klasy wewnętrzne:

Klasy zdefiniowane wewnątrz innej klasy.

```
class Zewnetrzna {
   class Wewnetrzna {
      void metoda() {
         System.out.println("Jestem klasą wewnętrzną");
      }
   }
}
```

6. Obsługa wyjątków, asercji

Obsługa wyjątków:

- **Try-catch:**

  ```
  try {
     int wynik = 10 / 0;
  } catch (ArithmeticException e) {
     System.out.println("Błąd: " + e.getMessage());
  }
  ```

- **Throws:** Deklaruje wyjątki, które metoda może zgłosić.

  ```
  public void metoda() throws IOException {
     throw new IOException("Błąd");
  }
  ```

Asercje:

Służą do weryfikacji założeń w kodzie.

```
assert liczba > 0 : "Liczba musi być dodatnia";
```

7. Zagadnienia związane z programowaniem generycznym

Generyki umożliwiają tworzenie klas, metod i interfejsów z parametrami typów.

```
class Generyczna<T> {
   private T element;

   public void setElement(T element) {
      this.element = element;
   }

   public T getElement() {
      return element;
   }
}
```

Przykład użycia:

```
Generyczna<String> obiekt = new Generyczna<>();
obiekt.setElement("Tekst");
System.out.println(obiekt.getElement());
```

8. Kolekcje w Java

Rodzaje kolekcji:

1. **Listy (List):**
   - Implementacje: ArrayList, LinkedList.
   - Przykład:

     ```
     List<String> lista = new ArrayList<>();
     lista.add("Element");
     ```

2. **Zbiory (Set):**
   - Implementacje: HashSet, TreeSet.
   - Przykład:

     ```
     Set<Integer> zbior = new HashSet<>();
     zbior.add(10);
     ```

3. **Mapy (Map):**
   - Implementacje: HashMap, TreeMap.
   - Przykład:

     ```
     Map<String, Integer> mapa = new HashMap<>();
     mapa.put("Klucz", 5);
     ```

wykorzystujemy Collections i **Stream API**.

1. Dodawanie elementów

Lista (List):

```
List<String> list = new ArrayList<>();
list.add("Element 1");
list.add("Element 2");
```

Zbiór (Set):

```
Set<Integer> set = new HashSet<>();
set.add(10);
set.add(20);
```

Mapa (Map):

```
Map<String, Integer> map = new HashMap<>();
map.put("Klucz1", 100);
map.put("Klucz2", 200);
```

2. Usuwanie elementów

Lista:

Usuwanie przez indeks:

```
list.remove(0); // Usuwa element na pozycji 0
```

Usuwanie przez wartość:

```
list.remove("Element 2");
```

Zbiór:

```
set.remove(10);
```

Mapa:

```
map.remove("Klucz1"); // Usuwa wpis na podstawie klucza
```

3. Modyfikowanie elementów

Lista:

Modyfikacja elementu na określonym indeksie:

```
list.set(0, "Nowy Element");
```

Mapa:

Zamiana wartości dla klucza:

```
map.put("Klucz2", 300);
```

4. Pobieranie elementów

Lista:

Pobranie elementu na podstawie indeksu:

```
String element = list.get(1);
```

Zbiór:

Nie ma dostępu przez indeks (iteracja):

```
for (Integer liczba : set) {
   System.out.println(liczba);
}
```

Mapa:

Pobranie wartości na podstawie klucza:

```
Integer wartosc = map.get("Klucz2");
```

5. Iterowanie po kolekcjach

Z użyciem pętli for-each:

Dla listy:

```
for (String elem : list) {
   System.out.println(elem);
}
```

Dla zbioru:

```
for (Integer liczba : set) {
   System.out.println(liczba);
}
```

Dla mapy:

```
for (Map.Entry<String, Integer> entry : map.entrySet()) {
   System.out.println(entry.getKey() + ": " + entry.getValue());}
```

Z użyciem Stream API:

```
list.stream().forEach(System.out::println);
```

6. Sprawdzanie zawartości

Lista:

```
boolean contains = list.contains("Element 1");
```

Zbiór:

```
boolean contains = set.contains(10);
```

Mapa:

Sprawdzanie istnienia klucza lub wartości:

```
boolean hasKey = map.containsKey("Klucz1");
boolean hasValue = map.containsValue(100);
```

7. Sortowanie

Lista:

Sortowanie naturalne:

```
Collections.sort(list);
```

Sortowanie z użyciem Comparatora:

```
list.sort(Comparator.reverseOrder());
```

Zbiór:

Zamiana na listę i sortowanie:

```
List<Integer> sortedList = new ArrayList<>(set);
Collections.sort(sortedList);
```

8. Filtracja i transformacja

Z użyciem **Stream API**:

Filtracja:

```
List<String> filteredList = list.stream()
                        .filter(s -> s.startsWith("E"))
                        .collect(Collectors.toList());
```

Transformacja:

```
List<String> upperCaseList = list.stream()
                         .map(String::toUpperCase)
                         .collect(Collectors.toList());
```

9. Łączenie i kopiowanie kolekcji

Łączenie list:

```
List<String> secondList = new ArrayList<>();
secondList.add("Element 3");
list.addAll(secondList);
```

Kopiowanie:

```
List<String> copiedList = new ArrayList<>(list);
```

10. Usuwanie elementów na podstawie warunku

Z użyciem **removeIf**:

```
list.removeIf(s -> s.startsWith("E"));
```

11. Wyczyszczenie kolekcji

Lista i Zbiór:

```
list.clear();
set.clear();
```

Mapa:

```
map.clear();
```

12. Rozmiar kolekcji

Lista i Zbiór:

java
Skopiuj kod

int size = list.size();

Mapa:

java
Skopiuj kod
int size = map.size();

1. Opisz zasadę działania Garbage Collectora JVM.
2. Opisz działanie maszyny JVM.
3. Opisz różnicę między ArrayList a LinkedList?
4. Czym się różni interfejs od klasy abstrakcyjnej?
5. Scharakteryzuj typy wyjątków i sposób ich obsługi?
6. Opisz charakterystykę programowania funkcyjnego i obiektowego oraz wymień różnice pomiędzy nimi.

1. Zasada działania Garbage Collectora JVM

Garbage Collector (GC) w JVM odpowiada za automatyczne zarządzanie pamięcią, usuwając obiekty, które nie są już używane przez aplikację.

Zasada działania:

1. **Generacje pamięci:** JVM dzieli pamięć na obszary:
    - **Young Generation:** Miejsce dla nowych obiektów. Składa się z podobszarów:
        - *Eden*: Nowe obiekty są tworzone tutaj.
        - *Survivor*: Obiekty, które przetrwają pewną liczbę cykli GC, są przenoszone tutaj.
    - **Old Generation (Tenured):** Dla długowiecznych obiektów.
    - **Metaspace:** Przechowuje dane klasy (od Javy 8 zastąpiło PermGen).
2. **Proces czyszczenia:**
    - **Mark-and-Sweep:** GC zaznacza obiekty, które są osiągalne (mark), a następnie usuwa te, które nie są (sweep).
    - **Compact:** Po usunięciu obiektów pamięć jest defragmentowana.
3. **Rodzaje GC:**
    - **Serial GC:** Prosty, jednowątkowy.
    - **Parallel GC:** Wielowątkowy, działa szybciej w systemach wieloprocesorowych.
    - **G1 GC:** Dzieli pamięć na regiony, efektywny dla dużych stert.

Zalety:

- Automatyczne zarządzanie pamięcią.
- Redukcja błędów, takich jak wycieki pamięci.

2. Działanie maszyny JVM

JVM (Java Virtual Machine) to wirtualna maszyna odpowiedzialna za uruchamianie kodu Java.

Etapy działania:

1. **Kompilacja do bajtkodu:** Kod źródłowy (.java) jest kompilowany do bajtkodu (.class) przez kompilator javac.
2. **Wczytanie klasy:** JVM ładuje bajtkod za pomocą ClassLoader.
3. **Weryfikacja:** Weryfikator sprawdza poprawność bajtkodu pod kątem bezpieczeństwa i zgodności.
4. **Kompilacja JIT (Just-In-Time):** JVM tłumaczy bajtkod na natywny kod maszynowy dla docelowej platformy w czasie wykonywania.
5. **Wykonywanie programu:** Silnik wykonawczy JVM uruchamia kod. JVM zarządza pamięcią (przydział i Garbage Collector), wątkami oraz obsługuje wyjątki.

Komponenty JVM:

- **ClassLoader:** Ładuje klasy do pamięci.
- **Execution Engine:** Odpowiada za wykonywanie kodu.
- **Heap:** Przechowuje obiekty.
- **Method Area:** Przechowuje metadane klas.
- **Stack:** Przechowuje ramki metod.

4. Różnica między ArrayList a LinkedList

Cecha	ArrayList	LinkedList
Struktura danych	Tablica dynamiczna	Lista dwukierunkowa
Czas dostępu	Szybki (O(1)) dla dostępu przez indeks	Wolniejszy (O(n))
Czas wstawiania/usuwania	Wolniejszy (O(n)) dla środka tablicy	Szybszy (O(1)) na początku/końcu
Zużycie pamięci	Mniejsze, brak wskaźników	Większe, przechowuje wskaźniki
Zastosowanie	Idealna do częstego odczytu	Idealna do częstych operacji wstawiania/usuwania

4. Różnica między interfejsem a klasą abstrakcyjną

Cecha	Interfejs	Klasa abstrakcyjna
Definicja	Zbiór metod abstrakcyjnych	Może zawierać zarówno metody abstrakcyjne, jak i zdefiniowane
Słowo kluczowe	interface	abstract class
Dziedziczenie	Klasa może implementować wiele interfejsów	Klasa może dziedziczyć tylko jedną klasę abstrakcyjną
Pola	Tylko stałe (static final)	Może zawierać zmienne instancji
Domyślne metody	Od Javy 8 można definiować metody default	Brak takich metod

5. Typy wyjątków i sposób ich obsługi

Typy wyjątków:

1. **Checked Exceptions:**
   - Muszą być obsłużone w kodzie.
   - Przykład: IOException, SQLException.
2. **Unchecked Exceptions:**
   - Wynikają z błędów programistycznych.
   - Przykład: NullPointerException, ArrayIndexOutOfBoundsException.
3. **Error:**
   - Powodują, że program przestaje działać.
   - Przykład: OutOfMemoryError.

Obsługa wyjątków:

1. **Try-catch:**

```
try {
   // Kod, który może zgłosić wyjątek
} catch (Exception e) {
   System.out.println("Błąd: " + e.getMessage());
}
```

2. **Finally:** Blok, który zawsze jest wykonywany.

```
try {
   // Kod
} finally {
   System.out.println("Zawsze wykonywane");
}
```

3. **Throws:** Przekazywanie wyjątków do wywołującego kodu.

```
public void metoda() throws IOException {
   throw new IOException("Błąd");
}
```

6. Charakterystyka programowania funkcyjnego i obiektowego

Programowanie obiektowe (OOP):

1. **Charakterystyka:**
   - Modeluje dane jako obiekty.
   - Główne cechy:
     - Enkapsulacja, dziedziczenie, polimorfizm.
   - Przykłady: Java, C++.
2. **Zalety:**
   - Modułowość i możliwość ponownego użycia kodu.
   - Łatwość modelowania rzeczywistych problemów.
3. **Wady:**
   - Złożoność w przypadku małych projektów.

Programowanie funkcyjne (FP):

1. **Charakterystyka:**
   - Opiera się na funkcjach jako podstawowych jednostkach przetwarzania danych.
   - Funkcje są czysto funkcyjne (bez efektów ubocznych).
   - Przykłady: Haskell, Scala, Java (od wersji 8).
2. **Zalety:**
   - Łatwość testowania i równoległości.
   - Brak efektów ubocznych.
3. **Wady:**
   - Trudność adaptacji dla osób przyzwyczajonych do OOP.

Różnice:

Cecha	OOP	FP
Podstawowy element	Obiekty	Funkcje
Stan danych	Stan jest zmienny	Dane są niemodyfikowalne
Paradygmat	Imperatywny	Deklaratywny
Dziedziczenie	Kluczowy mechanizm	Brak
Przykłady w Java	Klasy, obiekty	Wyrażenia lambda, strumienie

Relacyjne bazy danych

1. Podstawy modelu relacyjnego, diagramy ERD, normalizacja danych

Podstawy modelu relacyjnego

Model relacyjny to sposób organizacji danych w postaci tabel, które są powiązane relacjami. Każda tabela w modelu relacyjnym reprezentuje **relację** i składa się z:

- **Wierszy (rekordów)** – oznaczają pojedyncze wpisy.
- **Kolumn (atrybutów)** – określają cechy opisujące dane.

Kluczowe pojęcia:

- **Klucz podstawowy (Primary Key):** Unikalny identyfikator rekordu w tabeli (np. id).
- **Klucz obcy (Foreign Key):** Atrybut w jednej tabeli, który wskazuje na klucz podstawowy w innej tabeli.
- **Encje:** Obiekty, które mają cechy do przechowywania (np. Student, Kurs).
- **Atrybuty:** Właściwości encji (np. Imię, Nazwisko, Data urodzenia).

Diagramy ERD (Entity-Relationship Diagram)

Diagram ERD to graficzne przedstawienie modelu relacyjnego.

- **Elementy diagramu ERD:**
  - **Encje:** Reprezentowane przez prostokąty.
  - **Relacje:** Połączenia między encjami, oznaczane jako romby.
  - **Atrybuty:** Owalne symbole przypisane do encji.

  Przykład diagramu ERD:

- Encja Student z atrybutami: ID, Imię, Nazwisko.
- Encja Kurs z atrybutami: ID_Kursu, Nazwa.
- Relacja Zapisany_na między Student a Kurs.

Normalizacja danych

Normalizacja to proces organizacji danych w tabelach w celu wyeliminowania redundancji i zapewnienia spójności. Główne formy normalne to:

1. **Pierwsza forma normalna (1NF):**
   - Wszystkie wartości w tabeli są atomowe (brak list lub wielokrotnych wartości w jednej komórce).
2. **Druga forma normalna (2NF):**
   - Tabela spełnia 1NF i każdy atrybut, który nie jest częścią klucza podstawowego, zależy w pełni od tego klucza.
3. **Trzecia forma normalna (3NF):**
   - Spełnia 2NF i nie zawiera zależności przechodnich między atrybutami.

2. Podstawy SQL, funkcje agregujące SQL

Podstawy SQL

SQL (Structured Query Language) to język do zarządzania bazami danych. Podstawowe operacje to:

- **Tworzenie tabel:**

```
CREATE TABLE Student (
   ID INT PRIMARY KEY,
   Imię VARCHAR(50),
   Nazwisko VARCHAR(50)
);
```

- **Dodawanie danych:**

```
INSERT INTO Student (ID, Imię, Nazwisko)
VALUES (1, 'Jan', 'Kowalski');
```

- **Zapytania SELECT:**

```
SELECT * FROM Student;
```

- **Aktualizowanie danych:**

```
UPDATE Student
SET Nazwisko = 'Nowak'
WHERE ID = 1;
```

- **Usuwanie danych:**

```
DELETE FROM Student
```

```
WHERE ID = 1;
```

Funkcje agregujące SQL

Funkcje agregujące służą do obliczeń na zestawach danych. Przykłady:

- **SUM:** Suma wartości w kolumnie.

```
SELECT SUM(Salary) AS TotalSalary FROM Employees;
```

- **AVG:** Średnia wartość.

```
SELECT AVG(Age) AS AverageAge FROM Users;
```

- **COUNT:** Liczba rekordów.

```
SELECT COUNT(*) AS TotalStudents FROM Student;
```

- **MAX:** Maksymalna wartość.

```
SELECT MAX(Salary) AS HighestSalary FROM Employees;
```

- **MIN:** Minimalna wartość.

```
SELECT MIN(Salary) AS LowestSalary FROM Employees;
```

3. Łączenie tabel

Łączenie tabel pozwala pobierać dane z wielu tabel na podstawie relacji między nimi.

Rodzaje JOIN w SQL:

1. **INNER JOIN:**
    - Pobiera tylko te rekordy, które mają dopasowania w obu tabelach.

```
SELECT Student.Imię, Kurs.Nazwa
FROM Student
INNER JOIN Zapisany_na
ON Student.ID = Zapisany_na.ID_Studenta
INNER JOIN Kurs
ON Zapisany_na.ID_Kursu = Kurs.ID;
```

2. **LEFT JOIN (LEFT OUTER JOIN):**
    - Pobiera wszystkie rekordy z lewej tabeli, nawet jeśli nie mają dopasowań w prawej tabeli.

```
SELECT Student.Imię, Kurs.Nazwa
FROM Student
LEFT JOIN Zapisany_na
ON Student.ID = Zapisany_na.ID_Studenta;
```

3. **RIGHT JOIN (RIGHT OUTER JOIN):**

- Pobiera wszystkie rekordy z prawej tabeli, nawet jeśli nie mają dopasowań w lewej tabeli.

```
SELECT Kurs.Nazwa, Student.Imię
FROM Kurs
RIGHT JOIN Zapisany_na
ON Kurs.ID = Zapisany_na.ID_Kursu;
```

4. **FULL OUTER JOIN:**
   - Pobiera wszystkie rekordy z obu tabel, niezależnie od dopasowania.

```
SELECT Student.Imię, Kurs.Nazwa
FROM Student
FULL OUTER JOIN Zapisany_na
ON Student.ID = Zapisany_na.ID_Studenta;
```

Przykład łączenia tabel:

Tabela Student:

ID	Imię	Nazwisko
1	Jan	Kowalski
2	Anna	Nowak

Tabela Kurs:

ID	Nazwa
101	Matematyka
102	Informatyka

Tabela Zapisany_na:

ID_Studenta	ID_Kursu
1	101
2	102

Zapytanie:

```
SELECT Student.Imię, Kurs.Nazwa
FROM Student
INNER JOIN Zapisany_na ON Student.ID = Zapisany_na.ID_Studenta
INNER JOIN Kurs ON Zapisany_na.ID_Kursu = Kurs.ID;
```

Wynik:

Imię	Nazwa
Jan	Matematyka
Anna	Informatyka

1. Omów wady i zalety modelu relacyjnego.
2. Czym jest normalizacja i postacie normalne w kontekscie relacyjnych baz danych.
3. Zastosowania języka SQL.
4. Podział i omówienie typów kwerend w bazach danych.
5. W jakim celu używamy indeksy i klucze w relacyjnych bazach danych.
6. Opisz złączenia (JOIN) w SQL'u.

1. Omów wady i zalety modelu relacyjnego

Zalety modelu relacyjnego:

- **Łatwość zarządzania danymi:** Dane są przechowywane w tabelach w sposób uporządkowany, co ułatwia ich organizację.
- **Elastyczność:** Możliwość łączenia danych z różnych tabel za pomocą relacji.
- **Spójność danych:** Dzięki zastosowaniu kluczy i normalizacji minimalizuje się redundancję i zwiększa integralność danych.
- **Wysoka skalowalność:** Relacyjne bazy danych dobrze radzą sobie z dużą ilością danych i wieloma użytkownikami.
- **Wsparcie dla standardowego języka SQL:** Powszechne zastosowanie SQL pozwala na łatwe tworzenie, modyfikowanie i analizowanie danych.

Wady modelu relacyjnego:

- **Skomplikowana struktura:** Przy dużych systemach baza może stać się trudna w zarządzaniu, wymagając znacznych zasobów obliczeniowych.
- **Wydajność:** Przy dużej liczbie relacji i złożonych zapytań system może działać wolniej.
- **Brak wsparcia dla danych nienumerycznych i nieustrukturyzowanych:** Modele relacyjne nie są idealne dla danych takich jak multimedia, które lepiej obsługują bazy NoSQL.
- **Koszt implementacji:** Tworzenie relacyjnych baz danych wymaga znacznych zasobów w zakresie projektowania i administracji.

2. Czym jest normalizacja i postacie normalne w kontekście relacyjnych baz danych

Normalizacja:

Normalizacja to proces organizacji danych w bazach relacyjnych w celu:

- Redukcji redundancji (powtarzania danych).
- Zapewnienia spójności i integralności danych.
- Poprawy wydajności.

Postacie normalne:

1. **Pierwsza postać normalna (1NF):**
    - Wszystkie kolumny tabeli zawierają wartości atomowe (pojedyncze).
    - Brak powtarzających się grup danych.
2. **Druga postać normalna (2NF):**
    - Spełnia wymagania 1NF.
    - Każdy atrybut niekluczowy jest zależny od całego klucza podstawowego, a nie jego części (brak częściowych zależności).
3. **Trzecia postać normalna (3NF):**
    - Spełnia wymagania 2NF.
    - Nie zawiera zależności przechodnich, tzn. atrybuty niekluczowe nie zależą od innych atrybutów niekluczowych.
4. **Postać Boyce'a-Codda (BCNF):**
    - Bardziej rygorystyczna niż 3NF, każdy determinujący atrybut musi być kluczem.

3. Zastosowania języka SQL

SQL (Structured Query Language) jest używany do zarządzania danymi w relacyjnych bazach danych. Zastosowania obejmują:

- **Tworzenie baz danych i ich struktur:**
    - Definicja tabel, indeksów, widoków.
    - Komendy: CREATE, ALTER, DROP.
- **Operacje CRUD:**
    - **C**reate: Wstawianie danych (INSERT).
    - **R**ead: Pobieranie danych (SELECT).
    - **U**pdate: Aktualizacja danych (UPDATE).
    - **D**elete: Usuwanie danych (DELETE).
- **Zarządzanie użytkownikami i uprawnieniami:**
    - Przydzielanie praw dostępu (GRANT, REVOKE).
- **Agregacja i analiza danych:**
    - Obliczenia statystyczne, sortowanie i grupowanie.
- **Łączenie danych:**
    - Wykorzystanie złączeń (JOIN) do pracy z wieloma tabelami.

4. Podział i omówienie typów kwerend w bazach danych

Kwerendy (zapytania) można podzielić na kilka kategorii:

- **DML (Data Manipulation Language):**

    - Obsługuje manipulowanie danymi.
    - Komendy: SELECT, INSERT, UPDATE, DELETE.
- **DDL (Data Definition Language):**
    - Służy do definiowania struktury bazy danych.
    - Komendy: CREATE, ALTER, DROP.
- **DCL (Data Control Language):**
    - Kontroluje prawa dostępu do bazy danych.
    - Komendy: GRANT, REVOKE.
- **TCL (Transaction Control Language):**
    - Zarządza transakcjami.
    - Komendy: COMMIT, ROLLBACK, SAVEPOINT.

5. W jakim celu używamy indeksy i klucze w relacyjnych bazach danych

Indeksy:

- **Cel:**
    - Przyspieszenie wyszukiwania danych w tabelach.
    - Redukcja czasu wykonania zapytań.
- **Działanie:**
    - Tworzenie struktury danych podobnej do drzewa wyszukiwań, dzięki której baza szybciej znajduje odpowiednie rekordy.
- **Rodzaje indeksów:**
    - Indeksy unikalne.
    - Indeksy na jednej kolumnie lub wielu kolumnach.
    - Indeksy pełnotekstowe.

Klucze:

- **Klucz podstawowy (Primary Key):**
    - Gwarantuje unikalność rekordów w tabeli.
- **Klucz obcy (Foreign Key):**
    - Tworzy relacje między tabelami.
- **Indeks klucza:**
    - Tworzy domyślnie indeks na kolumnie będącej kluczem podstawowym, co przyspiesza wyszukiwanie.

6. Opisz złączenia (JOIN) w SQL'u

Złączenia w SQL umożliwiają pobieranie danych z więcej niż jednej tabeli na podstawie ich relacji.

Rodzaje złączeń:

1. **INNER JOIN:**
    - Zwraca rekordy, które mają dopasowania w obu tabelach.

```
SELECT A.column1, B.column2
FROM TableA A
INNER JOIN TableB B
```

```
ON A.common_field = B.common_field;
```

2. **LEFT JOIN (LEFT OUTER JOIN):**
   - Zwraca wszystkie rekordy z lewej tabeli oraz dopasowane rekordy z prawej tabeli (lub NULL, gdy brak dopasowania).

```
SELECT A.column1, B.column2
FROM TableA A
LEFT JOIN TableB B
ON A.common_field = B.common_field;
```

3. **RIGHT JOIN (RIGHT OUTER JOIN):**
   - Zwraca wszystkie rekordy z prawej tabeli oraz dopasowane rekordy z lewej tabeli (lub NULL, gdy brak dopasowania).

```
SELECT A.column1, B.column2
FROM TableA A
RIGHT JOIN TableB B
ON A.common_field = B.common_field;
```

4. **FULL OUTER JOIN:**
   - Zwraca wszystkie rekordy z obu tabel, nawet jeśli nie mają dopasowań.

```
SELECT A.column1, B.column2
FROM TableA A
FULL OUTER JOIN TableB B
ON A.common_field = B.common_field;
```

5. **CROSS JOIN:**
   - Tworzy iloczyn kartezjański obu tabel, łącząc każdy rekord z każdej tabeli.

```
SELECT A.column1, B.column2
FROM TableA A
CROSS JOIN TableB B;
```

6. **SELF JOIN:**
   - Tabela jest łączona sama ze sobą.

```
SELECT A.column1, B.column2
FROM TableA A, TableA B
WHERE A.column3 = B.column3;
```

Metody badawcze w informatyce i projektach inżynierskich

1. Metody, narzędzia i techniki badań naukowych

Metody badań naukowych:

- **Metoda empiryczna:**

    - Opiera się na obserwacji, eksperymentach i pomiarach.
    - Stosowana w naukach przyrodniczych, technicznych oraz społecznych.
    - Przykłady: badania terenowe, testy laboratoryjne.
- **Metoda teoretyczna:**
    - Analiza, synteza i modelowanie zjawisk na podstawie istniejącej wiedzy.
    - Stosowana w matematyce, filozofii, i naukach teoretycznych.
    - Przykłady: dedukcja, indukcja, analiza logiczna.
- **Metoda mieszana:**
    - Połączenie badań empirycznych i teoretycznych.
    - Przykład: modelowanie zjawiska na podstawie danych empirycznych.

Narzędzia badań:

- **Ankiety i kwestionariusze:** Służą do zbierania danych ilościowych lub jakościowych od respondentów.
- **Wywiady:** Pozwalają na pogłębioną analizę jakościową.
- **Eksperymenty:** Umożliwiają badanie związków przyczynowo-skutkowych.
- **Obserwacja:** Pozwala na rejestrowanie zjawisk w ich naturalnym środowisku.

Techniki badawcze:

- **Analiza statystyczna:** Umożliwia przetwarzanie danych liczbowych.
- **Analiza treści:** Badanie jakościowe treści dokumentów, wypowiedzi, publikacji.
- **Studia przypadków:** Głębokie badanie pojedynczego obiektu lub zjawiska.
- **Symulacje komputerowe:** Używane do modelowania i testowania w kontrolowanych warunkach.

2. Techniki analizy danych

Techniki ilościowe:

- **Statystyka opisowa:**
    - Obliczanie średniej, mediany, odchylenia standardowego.
    - Wizualizacja danych (wykresy, histogramy).
- **Analiza regresji:**
    - Badanie relacji między zmiennymi.
    - Przykłady: regresja liniowa, wielokrotna.
- **Testy statystyczne:**
    - Porównywanie grup, np. test t-Studenta, analiza wariancji (ANOVA).

Techniki jakościowe:

- **Analiza treści:**
    - Kategoryzowanie i kodowanie danych jakościowych.
- **Analiza narracyjna:**
    - Badanie historii lub opowieści respondentów.
- **Analiza fenomenologiczna:**
    - Zrozumienie doświadczeń badanych osób.

Techniki eksploracyjne:

- **Analiza skupień (clustering):**
  - Grupowanie danych w klastry na podstawie podobieństwa.
- **Metody redukcji wymiarowości:**
  - PCA (analiza głównych składowych) do uproszczenia dużych zbiorów danych.

3. Błędy badawcze i ich minimalizacja

Typy błędów badawczych:

- **Błąd systematyczny:**
  - Występuje, gdy metoda badawcza jest błędna.
  - Przykład: źle skalibrowany sprzęt pomiarowy.
  - **Minimalizacja:** Regularna kalibracja i standaryzacja metod.
- **Błąd losowy:**
  - Wynika z przypadkowych czynników wpływających na pomiar.
  - **Minimalizacja:** Zwiększenie liczby prób, zastosowanie średnich wyników.
- **Błąd próbkowania:**
  - Wynika z nieodpowiedniego doboru próby badawczej.
  - **Minimalizacja:** Losowy dobór próby, reprezentatywność próby.
- **Błąd pomiaru:**
  - Związany z niewłaściwym zastosowaniem narzędzi pomiarowych.
  - **Minimalizacja:** Szkolenie osób przeprowadzających badania.

Inne błędy:

- **Stronniczość badacza (bias):**
  - Może wpłynąć na interpretację wyników.
  - **Minimalizacja:** Blind testy, podwójne ślepe próby.
- **Błąd respondentów:**
  - Nieprawdziwe odpowiedzi w ankietach lub wywiadach.
  - **Minimalizacja:** Anonimowość respondentów.

4. Raportowanie wyników badań

Struktura raportu badawczego:

1. **Wstęp:**
   - Określenie celu i zakresu badania.
   - Przegląd literatury i uzasadnienie badania.
2. **Metodyka:**
   - Opis metod i narzędzi użytych w badaniu.
   - Charakterystyka próby badawczej.
3. **Wyniki:**
   - Prezentacja danych w formie tabel, wykresów, diagramów.
   - Wyniki analizy statystycznej lub jakościowej.
4. **Dyskusja:**
   - Interpretacja wyników w kontekście literatury i założeń badawczych.
   - Omówienie ograniczeń badania.
5. **Wnioski i rekomendacje:**
   - Kluczowe ustalenia.

    - Propozycje działań na podstawie wyników.
6. **Bibliografia:**
    - Wykaz literatury, z której korzystano.

Dobre praktyki w raportowaniu:

- **Przejrzystość i czytelność:** Użycie jasnego języka, unikanie zbędnego żargonu.
- **Weryfikowalność:** Prezentacja danych źródłowych i metodologii.
- **Obiektywizm:** Raportowanie zarówno pozytywnych, jak i negatywnych wyników.
- **Wizualizacja:** Wykorzystanie wykresów, tabel i schematów dla lepszego zrozumienia danych.

Formatowanie i styl:

- Zgodność z wytycznymi danego czasopisma, instytucji lub organizacji.
- Użycie narzędzi do zarządzania źródłami, np. Mendeley, Zotero.

Komputerowe wspomaganie zadań inżynierskich

1. Pojęcie skali i jej znaczenie w rysunkach technicznych

Skala to stosunek wymiarów przedstawionych na rysunku technicznym do rzeczywistych wymiarów obiektu. Określa, w jakiej proporcji rysunek obrazuje rzeczywiste wymiary przedmiotu.

- **Znaczenie skali:**
    - **Dokładność**: Skala pozwala na zachowanie proporcji, co jest kluczowe do wiernego odwzorowania przedmiotu w różnych rozmiarach.
    - **Przestronność**: Dzięki skali można zmieścić rysunek dużych obiektów, takich jak budynki czy maszyny, na kartce o standardowych rozmiarach.
    - **Przekonywalność**: Skala pozwala na dokładne przedstawienie detali, które mogą być niewidoczne w rzeczywistej wielkości obiektu.

Przykłady skal:

- **Skala 1:1** (rzeczywisty rozmiar) – obiekt na rysunku ma takie same wymiary jak w rzeczywistości.
- **Skala 1:2** – obiekt jest przedstawiony w połowie swojej rzeczywistej wielkości.
- **Skala 2:1** – obiekt jest przedstawiony dwukrotnie większy niż w rzeczywistości.

Skala jest określana na rysunku w specjalnej sekcji, często z użyciem symbolu lub liczby.

2. Podstawowe zasady tworzenia rysunku technicznego

Rysunek techniczny to wizualna reprezentacja obiektu, jego części, zespołów lub konstrukcji. Kluczowe zasady tworzenia rysunków technicznych to:

- **Zasada jednoznaczności**: Rysunek powinien być czytelny i jednoznacznie przedstawiać przedmiot bez potrzeby dodatkowych wyjaśnień. Wszystkie linie, oznaczenia i symbole muszą być dobrze zrozumiane przez odbiorcę.
- **Zasada dokładności**: Wszystkie wymiary i proporcje muszą być podane z wystarczającą dokładnością, aby rysunek mógł być użyty do wykonania obiektu.
- **Zasada prostoty**: Rysunki powinny być możliwie jak najprostsze, aby uniknąć zbędnych elementów, które mogą wprowadzać zamieszanie.
- **Zasada hierarchii informacji**: Rysunki techniczne powinny być zorganizowane w sposób, który ułatwia zrozumienie, z odpowiednim wykorzystaniem linii, wymiarów, opisów, oznaczeń itp.
- **Wykorzystanie odpowiednich widoków**: Zastosowanie różnych widoków, takich jak widok główny, boczny, przekroje, pozwala na lepsze zrozumienie detali konstrukcji.
- **Zasada stosowania norm i standardów**: Wiele branż posiada określone normy rysunkowe (np. ISO, ANSI), które określają, jak mają wyglądać rysunki techniczne.

3. Interfejs programu AutoCAD i jego podstawowe narzędzia

AutoCAD to jedno z najpopularniejszych narzędzi CAD do tworzenia rysunków 2D i 3D. Jego interfejs zawiera kilka podstawowych elementów:

- **Wstążka**: Górna część okna, która zawiera narzędzia do rysowania, edycji i wymiarowania. Jest podzielona na karty, które grupują różne funkcje (np. Home, Insert, Annotate).
- **Pasek poleceń**: Zwykle znajduje się na dole okna i pozwala na wprowadzanie komend oraz wskazówki w trakcie korzystania z programu.
- **Okno rysunku**: Główna przestrzeń robocza, w której użytkownik rysuje i edytuje obiekty.
- **Palety**: Narzędzia i właściwości obiektów, które można swobodnie przesuwać w interfejsie. Zawierają m.in. Style, Warstwy, Właściwości obiektów.
- **Linie poleceń**: Umożliwiają użytkownikowi wydawanie poleceń z klawiatury (np. LINE do rysowania linii).

Podstawowe narzędzia AutoCAD:

- **Rysowanie (Line, Circle, Arc)**: Narzędzia do rysowania podstawowych kształtów i obiektów.
- **Edycja (Trim, Extend, Offset, Move)**: Narzędzia do manipulacji obiektami.
- **Wymiarowanie**: Narzędzia do dodawania wymiarów i opisów na rysunku.
- **Warstwy (Layers)**: Pozwalają na zarządzanie obiektami na różnych warstwach, co ułatwia organizację rysunku.
- **Bloki (Blocks)**: Używane do tworzenia powtarzalnych elementów, które można łatwo wstawiać w różnych częściach rysunku.

4. Proces modelowania 3D w programie Autodesk Inventor

Autodesk Inventor to oprogramowanie CAD do tworzenia modeli 3D, wykorzystywane głównie w inżynierii mechanicznej. Proces modelowania 3D w Inventorze przebiega zazwyczaj w kilku krokach:

1. **Tworzenie szkicu 2D:**
    - Na początku projektant tworzy szkic 2D, który jest bazą do dalszego modelowania. Szkic jest rysowany na jednej z płaszczyzn roboczych (XY, XZ, YZ).
2. **Tworzenie operacji 3D:**
    - Na podstawie szkicu 2D użytkownik wykonuje operacje takie jak wyciąganie (Extrude), obracanie (Revolve), cięcie (Cut) i inne, które pozwalają na uzyskanie bryły 3D.
3. **Tworzenie komponentów i złożeń:**
    - Modele 3D są tworzone jako komponenty, które następnie mogą być złożone w większe zespoły. Do tworzenia złożeń używa się narzędzi do łączenia komponentów (np. pin, axle, mate).
4. **Analiza i testowanie:**
    - Po stworzeniu modelu 3D w Inventorze można przeprowadzać różne analizy, takie jak analiza wytrzymałościowa, analiza przepływu czy testowanie tolerancji.
5. **Rysunki techniczne:**
    - Na końcu procesu projektowania generowane są rysunki 2D na podstawie modelu 3D, które mogą być użyte do produkcji.

5. Korzyści i wyzwania związane z używaniem CAD w porównaniu do tradycyjnych metod rysowania technicznego

Korzyści:

- **Dokładność i precyzja**: Programy CAD pozwalają na dokładne odwzorowanie wymiarów i tolerancji, co może być trudne do osiągnięcia w tradycyjnych rysunkach.
- **Elastyczność**: Możliwość łatwej edycji i modyfikacji projektu, co jest bardzo czasochłonne w tradycyjnym rysowaniu.
- **Zarządzanie złożonymi projektami**: CAD pozwala na łatwe zarządzanie złożonymi projektami, zwłaszcza w zakresie dużych zespołów i systemów, gdzie liczne komponenty są ze sobą połączone.
- **Generowanie dokumentacji**: Automatyczne tworzenie rysunków technicznych i zestawów wymiarowych na podstawie modeli 3D.
- **Symulacje i analizy**: Możliwość przeprowadzania symulacji i testów przed produkcją, co pozwala na oszczędności i zmniejszenie ryzyka błędów produkcyjnych.

Wyzwania:

- **Wysokie koszty początkowe**: Programy CAD oraz odpowiedni sprzęt mogą wiązać się z dużymi kosztami początkowymi.
- **Krzywa uczenia się**: Użytkownicy muszą przejść przez proces nauki obsługi skomplikowanych narzędzi CAD, co może być czasochłonne.
- **Zależność od technologii**: Potrzebne są odpowiednie urządzenia komputerowe oraz oprogramowanie, które mogą być podatne na awarie lub wymagać regularnych aktualizacji.

1. Scharakteryzuj europejską metodę rzutowania prostokątnego.
2. Omów zasady sporządzania rysunku technicznego.
3. Zastosowanie i możliwości poszczególnych obszarów programu Autodesk Inventor Professional.

1. Scharakteryzuj europejską metodę rzutowania prostokątnego

Rzutowanie prostokątne jest jedną z podstawowych metod wykorzystywanych do przedstawiania obiektów 3D na płaszczyźnie 2D. Polega na rzutowaniu punktów obiektu na płaszczyznę rzutni wzdłuż linii prostopadłych do tej płaszczyzny. W kontekście **europejskiej metody rzutowania prostokątnego**, stosuje się system trzech podstawowych widoków: **widok frontowy (główny)**, **widok boczny** oraz **widok górny**.

- **Widok główny (frontalny)**: Reprezentuje obiekt z przodu i jest najważniejszym widokiem, który pokazuje główną geometrię przedmiotu. To jest widok, który jest najbardziej reprezentatywny w rysunku technicznym.
- **Widok górny**: Pokazuje obiekt z góry i może ujawniać szczegóły, które są niewidoczne w widoku głównym, takie jak wymiary w górnej części obiektu.
- **Widok boczny (prawy lub lewy)**: Przedstawia obiekt z boku, oferując dodatkowe informacje o jego głębokości i kształcie.

Zasady rysowania:

- Widoki te są rozmieszczone w ściśle określony sposób na rysunku, gdzie widok główny znajduje się zazwyczaj na górze, widok górny – poniżej, a widok boczny – obok widoku głównego.
- Wszystkie linie rzutni powinny być prostopadłe do płaszczyzny rzutni, co zapewnia, że perspektywa jest zachowana i nie zniekształca obrazu przedmiotu.

Zalety:

- Dokładność i jasność przedstawienia obiektu w trzech podstawowych widokach.
- Umożliwia szczegółowe ukazanie wymiarów oraz proporcji obiektów w różnych perspektywach.

2. Omów zasady sporządzania rysunku technicznego

Rysunek techniczny jest dokumentem wizualnym wykorzystywanym do przekazywania szczegółowych informacji o obiektach, konstrukcjach, maszynach, czy urządzeniach. Aby rysunek techniczny był czytelny i zrozumiały, musi spełniać kilka podstawowych zasad:

1. **Dokładność i precyzja**:
    - Rysunek musi odwzorowywać rzeczywiste wymiary i kształty obiektów z odpowiednią dokładnością.
    - Wszystkie wymiary muszą być podane w odpowiednich jednostkach miary (np. milimetry, metry) oraz muszą być dokładne, aby uniknąć błędów w produkcji.
2. **Zasady wymiarowania**:
    - Wymiary muszą być podane wyraźnie, z użyciem standardowych symboli i konwencji, takich jak strzałki, linie wymiarowe oraz jednostki miary.
    - Należy stosować odpowiednie skale, w tym przypadku na rysunku często stosuje się skalę 1:1 (rzeczywisty rozmiar) lub inną, aby dostosować rysunek do rozmiaru papieru.
3. **Czytelność**:
    - Rysunki muszą być wyraźne, z wykorzystaniem odpowiednich linii, takich jak linie ciągłe, przerywane, grube i cienkie, aby zrozumienie poszczególnych elementów było łatwe.

- Należy zachować odpowiednią przestrzeń pomiędzy widokami, aby nie zatarły się one ze sobą.

4. **Zasada jednoznaczności**:
   - Rysunek powinien być tak zaprezentowany, aby każda osoba, która go ogląda, była w stanie jednoznacznie zrozumieć przedstawiony przedmiot. W tym celu stosuje się powszechnie uznawane symbole, oznaczenia, oraz normy rysunkowe.
5. **Użycie odpowiednich widoków**:
   - Na rysunkach technicznych rysuje się różne widoki obiektu, takie jak widok główny, boczny, górny, a także przekroje, aby dokładnie pokazać elementy, które są niewidoczne z zewnątrz.
6. **Podpisy i adnotacje**:
   - Na rysunku technicznym umieszcza się podpisy, daty, nazwiska autorów, a także numery rysunków, które pomagają w identyfikacji dokumentu.
   - Często dodaje się również dodatkowe informacje w formie tekstów objaśniających (np. materiały, tolerancje, sposób montażu).

3. Zastosowanie i możliwości poszczególnych obszarów programu Autodesk Inventor Professional

Autodesk Inventor to zaawansowane oprogramowanie CAD służące do projektowania 3D, szczególnie w inżynierii mechanicznej. Posiada szereg funkcji, które umożliwiają tworzenie precyzyjnych modeli 3D, złożeń i dokumentacji technicznej. Oto główne obszary programu i ich możliwości:

1. **Modelowanie 3D (Part Design)**:
   - Umożliwia tworzenie pojedynczych części w przestrzeni 3D. Użytkownicy mogą rysować szkice, a następnie wyciągać, wycinać, obracać lub tworzyć inne operacje na tych szkicach, aby stworzyć bryły.
   - Modelowanie jest parametryczne, co pozwala na łatwą modyfikację wymiarów i kształtów obiektów.
2. **Złożenia (Assembly Design)**:
   - Umożliwia tworzenie złożeń, które składają się z wielu części. Części są ze sobą połączone za pomocą odpowiednich złączy (np. złącza, pręty, sworznie).
   - Możliwość testowania ruchu złożeń, wykrywania kolizji oraz obliczeń wytrzymałościowych.
3. **Tworzenie rysunków technicznych (Drawing)**:
   - Generowanie dokumentacji 2D na podstawie modeli 3D. Program umożliwia tworzenie rysunków technicznych z widokami, wymiarami, tolerancjami i innymi oznaczeniami.
   - Możliwość tworzenia widoków projekcyjnych, przekrojów, widoków ściśle dopasowanych do wymagań norm rysunkowych.
4. **Analizy i symulacje (Simulation)**:
   - Autodesk Inventor oferuje narzędzia do przeprowadzania symulacji, takich jak analiza wytrzymałościowa, analiza przepływów, symulacje termiczne czy analiza dynamiczna.
   - Dzięki tym narzędziom projektanci mogą testować, jak ich modele będą zachowywać się w rzeczywistych warunkach, co pozwala na wykrycie potencjalnych problemów przed rozpoczęciem produkcji.
5. **Tworzenie bloku i schematów (Sheet Metal Design)**:
   - Specjalne narzędzia do projektowania elementów wykonanych z blachy. Umożliwia tworzenie części blaszanych, które mogą być gięte lub cięte na maszynach CNC.

6. **Inżynieria odwrotna (Reverse Engineering)**:
    - Dzięki wykorzystaniu technologii skanowania 3D użytkownicy mogą importować dane skanów i na ich podstawie tworzyć modele CAD. To przydatne w przypadku, gdy brakuje pełnych danych o danym obiekcie.
7. **Rysowanie elektryczne i schematy (Electrical Design)**:
    - Autodesk Inventor posiada moduły umożliwiające projektowanie instalacji elektrycznych, co jest szczególnie przydatne w przypadku maszyn i urządzeń zasilanych elektrycznie.
8. **Integracja z innymi programami**:
    - Inventor pozwala na importowanie i eksportowanie modeli z innymi aplikacjami CAD, umożliwiając wymianę danych z programami jak AutoCAD, Revit, SolidWorks, a także integrację z systemami zarządzania danymi CAD (np. PDM).

Komunikacja i zarządzenie projektami

1. Definicja projektu i jego kluczowe cechy

Projekt to tymczasowe przedsięwzięcie, które ma na celu stworzenie unikalnego produktu, usługi lub rezultatu. Projekty są zdefiniowane przez określony cel, zakres, czas i zasoby, a ich realizacja wymaga planowania i koordynacji działań.

Kluczowe cechy projektu:

- **Tymczasowość:** Projekt ma określony czas trwania, który rozpoczyna się i kończy w określonym czasie. Zakończenie projektu jest osiągnięciem celu lub osiągnięciem zaplanowanego rezultatu.
- **Unikalność:** Rezultat projektu jest unikalny i różni się od innych działań w organizacji. Może to być nowy produkt, proces, usługa, czy inne rozwiązanie.
- **Cele:** Każdy projekt ma jasno określony cel lub zestaw celów, które mają zostać osiągnięte w trakcie jego realizacji.
- **Zakres:** Projekt ma określony zakres, który precyzuje, jakie działania i zadania są związane z projektem, a jakie nie.
- **Zasoby:** Projekty wymagają określonych zasobów (ludzkich, finansowych, technologicznych itp.) do realizacji zaplanowanych zadań.
- **Złożoność i ryzyko:** Projekty często wiążą się z ryzykiem, ponieważ ich realizacja jest związana z niepewnością, a sukces zależy od umiejętnego zarządzania ryzykiem i złożonością.

2. Cykl życia projektu

Cykl życia projektu odnosi się do faz, przez które przechodzi projekt, od jego inicjacji aż do zakończenia. Istnieje wiele modeli cyklu życia projektu, ale ogólnie można wyróżnić następujące etapy:

1. **Inicjacja**:
    - Na tym etapie projekt jest oficjalnie zatwierdzany i definiowany. Obejmuje to ustalenie ogólnych celów, zakresu oraz wymaganych zasobów.
    - Kluczowe działania: opracowanie wstępnego biznesplanu, analiza wykonalności, powołanie zespołu projektowego, ustalenie budżetu i harmonogramu.
2. **Planowanie**:

- Na tym etapie szczegółowo opracowywana jest strategia realizacji projektu. Określane są konkretne zadania, terminy, zasoby, a także potencjalne ryzyka.
- Kluczowe działania: tworzenie szczegółowego planu projektowego, ustalanie harmonogramu, alokacja zasobów, zarządzanie ryzykiem.

3. **Realizacja**:
   - To faza wykonawcza projektu, w której realizowane są zadania zgodnie z wcześniej opracowanym planem. To etap, w którym projekt nabiera realnych kształtów.
   - Kluczowe działania: wdrażanie planów, koordynowanie pracy zespołu, monitorowanie postępów, zarządzanie komunikacją, kontrola jakości.
4. **Monitorowanie i kontrola**:
   - Na tym etapie monitorowane są postępy w realizacji projektu, weryfikowane są różnice między zaplanowanymi a rzeczywistymi wynikami, a także podejmowane działania naprawcze w razie potrzeby.
   - Kluczowe działania: kontrolowanie budżetu, terminu, jakości oraz ryzyka, raportowanie postępów, dostosowywanie planów.
5. **Zakończenie**:
   - Po osiągnięciu celów projektu, projekt jest formalnie zakończony. Zrealizowane wyniki są przekazywane do użytku, zespół projektowy jest rozwiązany, a projekt jest zamykany.
   - Kluczowe działania: ocena projektu, zakończenie wszystkich działań, formalne zamknięcie, dokumentacja wyników, ocena skutków.

3. Metoda SMART w zarządzaniu celami projektowymi

Metoda **SMART** to technika używana do wyznaczania jasnych i mierzalnych celów. Skrót SMART oznacza:

- **S** (Specific – Specyficzny): Cel powinien być jasny i precyzyjny. Należy zdefiniować, co dokładnie ma zostać osiągnięte.
- **M** (Measurable – Mierzalny): Cel musi być mierzalny, aby można było ocenić, czy został osiągnięty. Określenie miar sukcesu jest kluczowe.
- **A** (Achievable – Osiągalny): Cel powinien być realistyczny i możliwy do osiągnięcia, uwzględniając dostępne zasoby i ograniczenia.
- **R** (Relevant – Istotny): Cel musi być istotny i mieć znaczenie dla ogólnych celów organizacji lub projektu.
- **T** (Time-bound – Określony w czasie): Cel powinien być określony w czasie, tzn. musi mieć określony termin realizacji.

Przykład SMART: „Zwiększyć sprzedaż produktu X o 20% w ciągu następnych 6 miesięcy w regionie Y”.

4. Analiza kosztów i korzyści projektu

Analiza kosztów i korzyści to technika używana do oceny ekonomicznej wykonalności projektu. Umożliwia ona zrozumienie, czy projekt przyniesie wystarczające korzyści w porównaniu do jego kosztów.

Etapy analizy kosztów i korzyści:

1. **Identyfikacja kosztów**: Wymaga określenia wszystkich kosztów związanych z projektem, takich jak koszty inwestycji początkowych, koszty operacyjne, koszty utrzymania i napraw, koszty personelu.
2. **Identyfikacja korzyści**: Należy zidentyfikować wszystkie potencjalne korzyści, zarówno materialne (np. zyski finansowe), jak i niematerialne (np. poprawa wizerunku firmy, zwiększenie efektywności).
3. **Porównanie kosztów z korzyściami**: Na tym etapie porównuje się całkowite koszty projektu z przewidywanymi korzyściami. W przypadku, gdy korzyści przewyższają koszty, projekt jest uznawany za wykonalny i opłacalny.
4. **Analiza rentowności**: Stosowanie takich wskaźników jak **netto wartość obecna (NPV)** czy **wewnętrzna stopa zwrotu (IRR)** pozwala na ocenę opłacalności inwestycji.

5. Ryzyko w projekcie i zarządzanie nim

Ryzyko w projekcie odnosi się do niepewności, która może wpłynąć na osiągnięcie celów projektu. Ryzyko może przybierać różne formy, takie jak ryzyko związane z budżetem, harmonogramem, jakością, technologią czy zespołem projektowym.

Etapy zarządzania ryzykiem:

1. **Identyfikacja ryzyk**: Określenie, które czynniki mogą stanowić zagrożenie dla projektu. Może to obejmować zmiany rynkowe, problemy technologiczne, niedobory zasobów czy zmieniające się regulacje prawne.
2. **Ocena ryzyk**: Po zidentyfikowaniu ryzyk, należy je ocenić pod kątem prawdopodobieństwa wystąpienia oraz potencjalnego wpływu na projekt. Do oceny ryzyk często używa się macierzy ryzyka.
3. **Planowanie reakcji na ryzyko**: Określenie działań mających na celu zminimalizowanie wpływu ryzyka na projekt. Można wybrać kilka strategii, takich jak:
    - **Unikanie ryzyka** – zmiana planu, aby zapobiec ryzyku.
    - **Łagodzenie ryzyka** – podjęcie działań w celu zmniejszenia wpływu ryzyka.
    - **Przenoszenie ryzyka** – przekazanie ryzyka innej stronie (np. ubezpieczenie).
    - **Akceptacja ryzyka** – przyjęcie ryzyka, jeśli jego koszt jest znikomy.
4. **Monitorowanie i kontrola ryzyka**: Na bieżąco monitoruje się ryzyka, analizuje skuteczność działań zarządzających i dostosowuje plany w razie zmiany sytuacji.

Zarządzanie ryzykiem pozwala zminimalizować negatywne skutki niepewności i poprawia szanse na sukces projektu.

Elektornika

1. Analiza obwodów prądu stałego z zastosowaniem prawa Ohma i praw Kirchhoffa

Prawo Ohma jest podstawowym prawem w analizie obwodów elektrycznych, które opisuje zależność między napięciem, prądem i oporem. Matematycznie jest wyrażone jako:

$V=I*R$

gdzie:

- V to napięcie (w woltach),
- I to natężenie prądu (w amperach),
- R to opór (w omach).

Prawo Ohma jest kluczowe przy analizie obwodów prądu stałego, gdzie prąd płynie w jednym kierunku. Obwody te składają się z elementów takich jak rezystory, źródła napięcia oraz różne połączenia tych elementów. Aby obliczyć prąd lub napięcie w takich obwodach, wykorzystujemy prawo Ohma oraz zasady łączenia elementów (szeregowo lub równolegle).

Prawa Kirchhoffa to dwa zasady, które są niezbędne przy rozwiązywaniu obwodów elektrycznych:

- **Prawo Kirchhoffa dla prądów:** Mówi, że suma prądów wpływających do węzła w obwodzie musi być równa sumie prądów wypływających z tego węzła. Mathematycznie: $\sum I_{in} = \sum I_{out}$
- **Prawo Kirchhoffa dla napięć:** Mówi, że suma spadków napięć w zamkniętej pętli obwodu musi wynosić zero. Oznacza to, że suma napięć w obwodzie elektrycznym (uwzględniając źródła napięcia i opory) w dowolnej zamkniętej pętli jest równa zeru: $\sum V = 0$

Analizując obwody prądu stałego, stosujemy zarówno prawo Ohma, jak i prawa Kirchhoffa, aby znaleźć wartości prądu, napięcia i mocy w różnych częściach obwodu, niezależnie od jego złożoności.

2. Elementy elektroniczne: zasada działania i przykłady zastosowania

W elektronice istnieje szeroki zakres elementów elektronicznych, które pełnią różne funkcje w obwodach.:

1. **Rezystor**:
    - **Zasada działania**: Rezystor ogranicza przepływ prądu w obwodzie zgodnie z prawem Ohma. Im większy opór, tym mniejszy prąd płynie przez dany element.
    - **Zastosowanie**: Stosowany do regulacji prądu, ochrony przed przepięciami, w dzielnikach napięcia, a także do ustawiania poziomów sygnału w obwodach analogowych i cyfrowych.
2. **Kondensator**:
    - **Zasada działania**: Kondensator przechowuje ładunek elektryczny i jego pojemność określa ilość ładunku, który może być zgromadzony przy danym napięciu.
    - **Zastosowanie**: Stosowane w obwodach filtrujących, wygładzających napięcie, w układach czasowych oraz jako elementy magazynujące energię.
3. **Dioda**:
    - **Zasada działania**: Dioda przepuszcza prąd tylko w jednym kierunku (zwykle w kierunku przewodzenia) i blokuje prąd w drugim kierunku (w kierunku zaporowym).
    - **Zastosowanie**: W prostownikach (przemiany prądu zmiennego na stały), ochronie obwodów (np. diody Zenera) oraz w układach przełączających.
4. **Tranzystor**:
    - **Zasada działania**: Tranzystor działa jako przełącznik lub wzmacniacz, gdzie mały prąd sterujący w jednym obwodzie (np. bazie) może kontrolować dużą moc w innym obwodzie (np. kolektor-emiter).

- **Zastosowanie**: Wzmacniacze sygnałów, przełączniki, układy logiczne, konwersja sygnałów analogowych na cyfrowe.

5. **Cewka (indukcyjność)**:
   - **Zasada działania**: Cewka przechowuje energię w postaci pola magnetycznego, które generuje podczas przepływu prądu. Wytwarza indukcyjne opóźnienie zmiany prądu.
   - **Zastosowanie**: W obwodach filtrujących, w układach resonansowych, w transformatorach oraz w układach magazynujących energię.

6. Bateria:

- **Zasada działania:** Bateria to źródło energii elektrycznej, które wytwarza napięcie przez reakcję chemiczną między materiałami elektrody. Baterie mogą być pierwotne (jednorazowe) lub wtórne (wielokrotnego ładowania). Podczas rozładowywania elektrody reagują z elektrolitem, uwalniając elektrony, które przepływają przez obwód zewnętrzny, dostarczając energię.
- **Zastosowanie:** Baterie są powszechnie używane jako przenośne źródła energii w urządzeniach takich jak telefony komórkowe, piloty, zegarki, latarki, a także w urządzeniach medycznych i urządzeniach elektronicznych. Baterie pierwotne (np. alkaliczne) są jednorazowe, natomiast baterie wtórne (np. litowo-jonowe) są wielokrotnego użytku i stosowane w akumulatorach.

7. Akumulator:

- **Zasada działania:** Akumulator to rodzaj baterii, która może być wielokrotnie ładowana i rozładowywana. Działa na zasadzie reakcji chemicznych, które zachodzą między elektrodami i elektrolitem. W procesie ładowania akumulatora energia elektryczna przekształca się w energię chemiczną, a w trakcie rozładowania energia chemiczna przekształca się z powrotem w energię elektryczną. Akumulatory mogą pracować przy wyższych natężeniach prądu niż baterie.
- **Zastosowanie:** Akumulatory są szeroko wykorzystywane w pojazdach elektrycznych (np. akumulatory samochodowe), urządzeniach przenośnych (np. laptopy, telefony komórkowe), a także w energii odnawialnej (np. do magazynowania energii z paneli słonecznych). Akumulatory litowo-jonowe są najczęściej wykorzystywane w nowoczesnych urządzeniach przenośnych.

3. Właściwości i analiza obwodów prądu przemiennego

Obwody prądu przemiennego (AC) charakteryzują się tym, że napięcie i prąd zmieniają swój kierunek i wartość w czasie. Obwody te różnią się od obwodów prądu stałego (DC), w których prąd płynie w jednym kierunku.

Podstawowe właściwości prądu przemiennego:

- **Fala sinusoidalna**: Prąd i napięcie zmieniają swoje wartości w sposób sinusoidalny, co oznacza, że mają określoną amplitudę, częstotliwość i fazę.
- **Amplituda**: Maksymalna wartość napięcia lub prądu w cyklu.
- **Częstotliwość (f)**: Liczba cykli na sekundę (w jednostkach Hz).
- **Okres (T)**: Czas potrzebny do zakończenia jednego pełnego cyklu.
- **Faza**: Określa, w jakim punkcie cyklu znajduje się dany sygnał.

Analiza obwodów prądu przemiennego:

- **Rezystancja (R)**: W obwodach prądu przemiennego rezystancja zachowuje się tak samo jak w obwodach prądu stałego.
- **Reaktancja indukcyjna (XL)**: W cewkach występuje opór dla zmian prądu, który rośnie wraz z częstotliwością prądu przemiennego. Jest wyrażany jako $X_L=2\pi fL$, gdzie f to częstotliwość, a L to indukcyjność.
- **Reaktancja pojemnościowa (XC)**: W kondensatorach opór dla zmian napięcia zależy od częstotliwości i jest wyrażany jako $X_C=1/(2\pi fC)$, gdzie f to częstotliwość, a C to pojemność.

W analizie obwodów prądu przemiennego ważne jest uwzględnienie zarówno oporu (rezystancji), jak i reaktancji (indukcyjnej i pojemnościowej), które razem tworzą impedancję Z, która jest odpowiednikiem oporu w obwodach prądu przemiennego.

4. Układy cyfrowe i logika cyfrowa

Układy cyfrowe to układy elektroniczne, które operują na sygnałach binarnych (0 i 1), a ich zadaniem jest przetwarzanie informacji w postaci cyfrowej. Są one wykorzystywane w komputerach, mikrokontrolerach, systemach komunikacyjnych i innych urządzeniach elektronicznych.

Logika cyfrowa to dziedzina zajmująca się projektowaniem układów cyfrowych przy użyciu operacji logicznych. Podstawowe operacje logiczne to:

- **AND** (iloczyn): Wynik jest równy 1 tylko wtedy, gdy oba wejścia są równe 1.
- **OR** (suma): Wynik jest równy 1, gdy przynajmniej jedno z wejść jest równe 1.
- **NOT** (negacja): Wynik jest odwrotnością wejścia, czyli 0 staje się 1, a 1 staje się 0.
- **XOR** (suma modulo 2): Wynik jest równy 1 tylko wtedy, gdy jedno z wejść jest 1, ale nie oba.

Brama logiczna to podstawowy element układu cyfrowego, realizujący jedną z operacji logicznych. Przykładowe bramy:

- **Brama AND**: Realizuje funkcję logiczną AND.
- **Brama OR**: Realizuje funkcję logiczną OR.
- **Brama NOT**: Realizuje funkcję logiczną NOT.
- **Brama NAND, NOR, XOR, XNOR**: Są to bardziej złożone operacje, których działanie jest odwrotnością standardowych bram logicznych (np. NAND to AND negowane).

Układy kombinacyjne i sekwencyjne:

- **Układy kombinacyjne**: Wyjście zależy tylko od aktualnych wartości wejść (np. sumatory, mnożniki, dekodery).
- **Układy sekwencyjne**: Wyjście zależy od bieżących wejść i wcześniejszych stanów (np. przerzutniki, liczniki, rejestry).

1. Prawo Ohma, Prawa Kirchoffa.
2. Elektronika analogowa – elementy RLC.

3. Budowa oraz zasada działania tranzystora bipolarnego, który jest wzmacniaczem i . może jednocześnie być przełącznikiem.

4. Zasada działania i zastosowanie komparatora napięcia.

5. Dlaczego we wzmacniaczach operacyjnych jest stosowane napięciowe ujemne sprzężenie zwrotne?

1. Prawo Ohma, Prawa Kirchhoffa

Prawo Ohma: Prawo Ohma jest podstawowym prawem w analizie obwodów elektrycznych. Określa ono zależność między napięciem (V), natężeniem prądu (I) i oporem (R) w obwodzie elektrycznym. Jest ono wyrażone równaniem:

$V=I*R$

gdzie:

- V to napięcie (w woltach),
- I to natężenie prądu (w amperach),
- R to opór (w omach).

Prawo Ohma jest podstawą przy obliczaniu prądu, napięcia oraz oporu w obwodach elektrycznych.

Prawa Kirchhoffa:

1. **Prawo Kirchhoffa dla prądów** Mówi, że suma prądów wpływających do węzła w obwodzie elektrycznym musi być równa sumie prądów wypływających z tego węzła. Przykład: jeśli do węzła wpływają prądy I_1 i I_2, a wypływają I_3, to: $I_1+I_2=I_3$
2. **Prawo Kirchhoffa dla napięć** Mówi, że suma napięć w zamkniętej pętli obwodu elektrycznego jest równa zeru. Oznacza to, że suma napięć źródeł i spadków napięć w elementach pasywnych (rezystory, cewki, kondensatory) w tej pętli wynosi zero: $\sum V=0$

2. Elektronika analogowa – elementy RLC

Elementy **RLC** to podstawowe komponenty używane w elektronice analogowej: **rezystory (R)**, **induktory (L)** i **kondensatory (C)**. Każdy z tych elementów pełni unikalną rolę w obwodach analogowych, szczególnie w filtrach, oscylatorach, wzmacniaczach oraz układach czasowych.

- **Rezystor (R)**: Jest to element, który ogranicza przepływ prądu. Przestraja napięcie, zgodnie z prawem Ohma, i jest używany w wielu obwodach do ustalania poziomów napięcia i prądu. Wartość rezystora wyraża się w omach (Ω).
- **Induktor (L)**: Jest to element, który przechowuje energię w postaci pola magnetycznego. Induktor przeciwdziała zmianom prądu, tzn. stara się utrzymać stały prąd, a jego wartość wyraża się w henrach (H). W obwodach prądu przemiennego (AC) induktor ma właściwości reaktancji indukcyjnej, zależnej od częstotliwości sygnału.

- **Kondensator (C)**: Kondensator przechowuje energię w postaci pola elektrycznego. Jego funkcją jest przechowywanie ładunku elektrycznego, co wprowadza opóźnienia w zmianach napięcia w obwodzie. Kondensatory mają właściwości reaktancji pojemnościowej, zależnej od częstotliwości sygnału. Ich pojemność wyrażana jest w faradach (F).

Elementy RLC są często łączone w układach filtrujących, gdzie wpływają na częstotliwość sygnału. Zestawienie tych elementów pozwala na tworzenie filtrów dolnoprzepustowych, górnoprzepustowych, pasmowoprzepustowych i pasmowozaporowych.

3. Budowa oraz zasada działania tranzystora bipolarnego

Tranzystor bipolarny jest jednym z najważniejszych elementów półprzewodnikowych w elektronice. Tranzystory bipolarnie dzielą się na dwa główne typy: **NPN** i **PNP**. Składają się z trzech warstw materiału półprzewodnikowego: emiter, baza i kolektor.

- **Budowa**: Tranzystor bipolarny składa się z trzech warstw półprzewodnikowych: warstwa typu N (emiter), warstwa typu P (baza) i ponownie warstwa typu N (kolektor) dla tranzystora NPN, lub odwrotnie dla tranzystora PNP.
- **Zasada działania**:
    - W tranzystorze NPN, gdy na bazę przyłożone jest odpowiednie napięcie (względem emitera), powoduje to przepływ prądu z emitera do kolektora. Mały prąd bazy steruje dużym prądem kolektora, co umożliwia wzmocnienie sygnału.
    - Tranzystor może działać zarówno jako **wzmacniacz** (zwiększając moc sygnału wejściowego), jak i **przełącznik** (włączając lub wyłączając obwód).

4. Zasada działania i zastosowanie komparatora napięcia

Komparator napięcia to układ elektroniczny, którego zadaniem jest porównanie dwóch napięć wejściowych i generowanie sygnału wyjściowego, który wskazuje, które z napięć jest wyższe.

- **Zasada działania**: Komparator napięcia nie ma wzmocnienia jak wzmacniacz operacyjny i działa jak przełącznik: jeśli napięcie na jednym wejściu jest większe od napięcia na drugim, komparator wytwarza określoną wartość napięcia wyjściowego (np. 0V lub napięcie zasilania). Może być to np. stan wysoki (1) lub stan niski (0).
- **Zastosowanie**: Komparatory są powszechnie wykorzystywane w systemach, które wymagają detekcji progów napięcia, takich jak:
    - Przełączanie stanów w systemach cyfrowych.
    - Układy detekcji napięcia.
    - Zabezpieczenia przed przepięciami.

5. Dlaczego we wzmacniaczach operacyjnych stosowane jest napięciowe ujemne sprzężenie zwrotne?

Napięciowe ujemne sprzężenie zwrotne w wzmacniaczach operacyjnych jest stosowane w celu:

- **Stabilizacji wzmocnienia**: Sprzężenie zwrotne zmienia charakterystykę wzmacniacza, co pozwala na ustalenie dokładnego wzmocnienia. Dzięki temu wyjście wzmacniacza jest bardziej stabilne, a wartość wzmocnienia nie zależy od zmieniających się parametrów tranzystorów.
- **Poprawy pasma przenoszenia**: Ujemne sprzężenie zwrotne zmniejsza nieliniowość wzmocnienia, co pozwala na uzyskanie szerszego pasma przenoszenia.
- **Zwiększenie liniowości i redukcję zniekształceń**: Zastosowanie sprzężenia zwrotnego zmniejsza nieliniowe zniekształcenia w wzmocnionym sygnale, co poprawia jakość sygnału wyjściowego.
- **Kontrola wydajności energetycznej**: Ujemne sprzężenie zwrotne pozwala na dostosowanie wzmacniacza do różnych warunków obciążenia, umożliwiając jego efektywniejsze działanie.

Sprzężenie zwrotne w wzmacniaczach operacyjnych poprawia ich dokładność, stabilność i możliwości regulacji wzmocnienia, co czyni je niezwykle użytecznymi w wielu zastosowaniach, takich jak filtry, wzmacniacze sygnałów, konwertery i układy analogowe.

Administracja i integracja systemów operacyjnych

1. RAID – Budowa, porównanie właściwości różnych typów, główne cechy jakościowe

RAID (Redundant Array of Independent Disks) to technologia, która łączy kilka fizycznych dysków twardych w jeden logiczny dysk w celu zwiększenia wydajności, niezawodności lub obu tych właściwości. RAID może być wykorzystywany zarówno w systemach serwerowych, jak i w urządzeniach do przechowywania danych. Istnieje wiele poziomów RAID, każdy z innymi właściwościami.

Budowa RAID

RAID składa się z co najmniej dwóch dysków twardych, które są połączone w konfigurację, umożliwiającą wspólne przetwarzanie danych. W zależności od wybranego poziomu RAID, dane mogą być rozdzielane (striped), replikowane (mirrored) lub rozdzielane i replikowane w różny sposób.

Poziomy RAID

1. **RAID 0 – Striping**:
    - **Budowa**: Dane są dzielone na bloki i zapisywane na wielu dyskach (zwykle 2). Dzięki temu zwiększa się wydajność zapisu i odczytu.
    - **Zalety**: Wysoka wydajność, zwiększenie szybkości odczytu i zapisu.
    - **Wady**: Brak redundancji – awaria jednego dysku prowadzi do utraty danych.

    - **Zastosowanie**: W systemach wymagających wysokiej wydajności, gdzie niezawodność nie jest priorytetem (np. w edycji wideo).
2. **RAID 1 – Mirroring**:
    - **Budowa**: Dane są kopiowane na dwa lub więcej dysków. Każdy dysk zawiera kopię tych samych danych.
    - **Zalety**: Wysoka niezawodność, ponieważ dane są duplikowane.
    - **Wady**: Wydajność nie jest tak wysoka jak w RAID 0, ponieważ zapis jest wykonywany na dwóch dyskach jednocześnie.
    - **Zastosowanie**: Idealny do przechowywania ważnych danych, gdzie niezawodność jest kluczowa.
3. **RAID 5 – Striping z parzystością**:
    - **Budowa**: Dane są dzielone na bloki i zapisywane na różnych dyskach, a dodatkowo dołączana jest informacja o parzystości, która pozwala na odbudowę danych w przypadku awarii jednego dysku.
    - **Zalety**: Dobre połączenie wydajności i redundancji, minimalizacja przestrzeni zajmowanej przez dane.
    - **Wady**: Niższa wydajność zapisu w porównaniu do RAID 0, ponieważ każdy zapis wymaga obliczeń parzystości.
    - **Zastosowanie**: W systemach, gdzie zarówno wydajność, jak i bezpieczeństwo danych są istotne, np. w serwerach baz danych.
4. **RAID 6 – Podobne do RAID 5, ale z dwiema parzystościami**:
    - **Budowa**: Podobnie jak RAID 5, ale z dodatkowymi blokami parzystości, co umożliwia odbudowę danych, nawet jeśli dwa dyski ulegną awarii.
    - **Zalety**: Bardzo wysoka niezawodność, możliwość odbudowy danych przy awarii dwóch dysków.
    - **Wady**: Niższa wydajność zapisu i większa utrata przestrzeni w porównaniu do RAID 5.
    - **Zastosowanie**: W krytycznych środowiskach, gdzie niezawodność jest absolutnie kluczowa.
5. **RAID 10 (1+0)**:
    - **Budowa**: Jest połączeniem RAID 1 i RAID 0. Dane są zarówno kopiowane (RAID 1), jak i dzielone na wiele dysków (RAID 0).
    - **Zalety**: Wysoka wydajność i redundancja, idealne połączenie szybkości i bezpieczeństwa.
    - **Wady**: Wymaga przynajmniej czterech dysków i zajmuje więcej przestrzeni w porównaniu do innych poziomów RAID.
    - **Zastosowanie**: W systemach wymagających dużej wydajności i wysokiego poziomu niezawodności.

Główne cechy jakościowe RAID:

- **Wydajność**: Niektóre poziomy RAID (np. RAID 0) oferują wyższą wydajność, podczas gdy inne (np. RAID 5, RAID 6) koncentrują się na zapewnieniu redundancji kosztem nieco niższej wydajności.
- **Niezawodność**: RAID 1, RAID 5, RAID 6 i RAID 10 oferują różne poziomy redundancji, zapewniając ochronę przed awarią dysku.
- **Koszt**: RAID 1 i RAID 10 wymagają więcej dysków, co zwiększa koszt przestrzeni dyskowej, podczas gdy RAID 0 jest tańszy, ale nie zapewnia żadnej redundancji.

2. LVM – Zasada działania, zastosowanie

LVM (Logical Volume Management) to system zarządzania dyskami, który umożliwia tworzenie elastycznych, łatwych do zarządzania przestrzeni dyskowych w systemach Linux. Dzięki LVM można tworzyć **logiczne wolumeny**, które mogą być dynamicznie zmieniane, powiększane lub zmniejszane, bez konieczności przeprowadzania operacji na fizycznych dyskach.

Zasada działania:

- **Punkty montowania**: LVM pozwala na połączenie różnych fizycznych dysków w jedną dużą jednostkę logiczną.
- **Punkty logiczne**: LVM pozwala na tworzenie wolumenów logicznych (LV – Logical Volumes) na grupach wolumenów (VG – Volume Groups), które mogą obejmować wiele fizycznych dysków.
- **Grupy wolumenów**: LVM łączy przestrzenie dyskowe z różnych dysków w jedną grupę, umożliwiając łatwiejsze zarządzanie przestrzenią.
- **Elastyczność**: LVM pozwala na rozbudowę przestrzeni dyskowej bez przestojów. Można dodać nowe dyski, powiększać istniejące wolumeny logiczne, a także zmieniać rozmiary i tworzyć nowe wolumeny.

Zastosowanie:

- **Elastyczne zarządzanie przestrzenią dyskową**: LVM ułatwia dodawanie nowych dysków, rozdzielanie i przydzielanie przestrzeni, co czyni zarządzanie dyskami prostszym.
- **Tworzenie migawkowych kopii zapasowych (snapshots)**: Możliwość tworzenia kopii systemów plików na wolumenach logicznych.
- **Zarządzanie przestrzenią w centrach danych**: LVM jest szeroko stosowany w środowiskach serwerowych i centrach danych, gdzie wymagana jest elastyczność i łatwość w zarządzaniu dużymi przestrzeniami dyskowymi.

3. Podstawowe zasady bezpiecznego administrowania systemem poprzez zdalny dostęp przez SSH

SSH (Secure Shell) jest protokołem umożliwiającym bezpieczne połączenie z systemem zdalnym, zapewniając szyfrowanie i uwierzytelnianie. Zdalne administrowanie systemem za pomocą SSH wymaga stosowania kilku zasad, które zapewnią bezpieczeństwo operacji.

Zasady:

1. **Silne hasła**:
    - Używanie długich, trudnych do odgadnięcia haseł lub preferowanie uwierzytelniania za pomocą kluczy SSH, które są znacznie bezpieczniejsze.
2. **Używanie kluczy SSH zamiast haseł**:
    - Klucze SSH są bardziej bezpieczne niż tradycyjne hasła, ponieważ wykorzystują pary kluczy publiczny/prywatny. Klucz prywatny powinien być przechowywany w bezpiecznym miejscu.
3. **Ograniczenie dostępu tylko do zaufanych adresów IP**:
    - Warto skonfigurować firewall, aby zezwalać na połączenia SSH tylko z określonych adresów IP, które są zaufane. Można to zrobić w pliku konfiguracyjnym SSH.
4. **Zmienianie portu domyślnego SSH (port 22)**:

- Aby zmniejszyć ryzyko ataków typu brute force, warto zmienić domyślny port SSH na mniej popularny.

5. **Używanie dwuskładnikowego uwierzytelniania (2FA)**:
   - Dodatkowe warstwy uwierzytelniania (np. kod SMS lub aplikacja mobilna) mogą znacznie poprawić bezpieczeństwo.
6. **Używanie konfiguracji `PermitRootLogin no`**:
   - Warto wyłączyć logowanie się bezpośrednio na konto root. Zamiast tego zaleca się logowanie się jako zwykły użytkownik i używanie `sudo`.
7. **Monitoring i logi**:
   - Regularne monitorowanie logów połączeń SSH i używanie narzędzi do detekcji włamań pozwala na szybsze wykrycie prób ataków.
8. **Aktualizowanie oprogramowania**:
   - Regularne aktualizowanie systemu operacyjnego i oprogramowania SSH zapewnia, że wszystkie luki w zabezpieczeniach są załatane na bieżąco.

Integracja sieci i usług

1. Podstawy routingu i główne protokoły routingu (np. RIP, OSPF, BGP)

Routing to proces określania ścieżki, którą pakiety danych powinny podróżować w sieci komputerowej, aby dotrzeć do docelowego adresu. W ramach routingu, routery analizują tabelę routingu i decydują, które z dostępnych ścieżek są najodpowiedniejsze dla danego pakietu.

Główne protokoły routingu:

1. **RIP (Routing Information Protocol)**:
   - **Opis**: Jest to protokół wektora odległości, który ustala najkrótszą ścieżkę na podstawie liczby przeskoków (hopów). RIP wysyła okresowo pełne informacje o swojej tabeli routingu.
   - **Zalety**: Prosty w implementacji, działa na małych i średnich sieciach.
   - **Wady**: Mało skalowalny i mało efektywny w dużych sieciach, brak obsługi większych sieci (np. OSPF).
2. **OSPF (Open Shortest Path First)**:
   - **Opis**: Protokół typu link-state, który używa algorytmu Dijkstry do wyboru najkrótszej ścieżki na podstawie stanu łączy między routerami. OSPF jest bardziej skomplikowany niż RIP, ale bardziej wydajny i skalowalny w dużych sieciach.
   - **Zalety**: Szybkie przywracanie po awarii, efektywniejszy niż RIP w dużych i rozproszonych sieciach.
   - **Wady**: Wymaga więcej zasobów i jest bardziej złożony w konfiguracji.
3. **BGP (Border Gateway Protocol)**:
   - **Opis**: Protokół wymiany informacji routingu między różnymi systemami autonomicznymi (AS), czyli dużymi sieciami. BGP jest podstawowym protokołem używanym w Internecie do routingu między routerami w różnych sieciach.
   - **Zalety**: Skalowalność i elastyczność, umożliwia wymianę informacji o trasach między różnymi sieciami.
   - **Wady**: Złożony i wymaga zaawansowanej konfiguracji. BGP jest także podatny na ataki, takie jak ataki typu prefix hijacking.

2. Algorytmy routingu i ich zastosowania w zarządzaniu ruchem sieciowym

Algorytmy routingu są kluczowe w procesie zarządzania ruchem w sieci, ponieważ decydują o najefektywniejszym kierowaniu pakietów danych. Najpopularniejsze algorytmy to:

1. **Algorytm Bellmana-Forda**:
    - **Zastosowanie**: Używany w protokołach typu wektor odległości (np. RIP). W tym algorytmie każdy router przechowuje informacje o najlepszej znanej trasie i wymienia się tymi informacjami z sąsiednimi routerami.
    - **Zalety**: Prostota implementacji.
    - **Wady**: Może prowadzić do problemów związanych z tzw. "count-to-infinity", co oznacza, że algorytm nie jest wystarczająco szybki do szybkiej reakcji na zmiany w sieci.
2. **Algorytm Dijkstry**:
    - **Zastosowanie**: Używany w protokołach typu link-state, takich jak OSPF. Algorytm ten oblicza najkrótszą trasę do celu, uwzględniając wagę łączy.
    - **Zalety**: Szybsza i bardziej wydajna reakcja na zmiany w sieci.
    - **Wady**: Większe wymagania zasobów, ponieważ router musi znać pełny stan sieci.
3. **Algorytmy oparte na tablicach routingu**:
    - **Zastosowanie**: Routery przechowują tabele routingu, które są następnie wykorzystywane do wybierania najlepszych tras.
    - **Zalety**: Umożliwiają szybkie przetwarzanie informacji o trasach.
    - **Wady**: Skalowalność jest problemem w większych sieciach.

3. Sieci VLAN

VLAN (Virtual Local Area Network) to technologia, która umożliwia podział jednej fizycznej sieci LAN na kilka logicznych sieci, co pozwala na segmentację ruchu sieciowego i poprawę zarządzania ruchem.

Zasady działania VLAN:

- **Izolacja**: Urządzenia w różnych VLANach są odseparowane od siebie, mimo że mogą być podłączone do tej samej fizycznej sieci.
- **Tworzenie VLANów**: Sieć może być podzielona na różne grupy (np. departamenty firmy), co pozwala na lepszą organizację i bezpieczeństwo.
- **Przekazywanie danych**: W VLANie urządzenia mogą komunikować się tylko z innymi urządzeniami w tej samej sieci VLAN, chyba że skonfigurowane są specjalne połączenia (np. routery między VLANami).

Zalety VLAN:

- Zwiększenie bezpieczeństwa, ponieważ urządzenia z różnych VLANów są odizolowane.
- Lepsza kontrola nad ruchem i mniejsze zagęszczenie w sieci.
- Zwiększenie wydajności, ponieważ ruch jest ograniczany do wybranych segmentów sieci.

4. Bezpieczeństwo protokołów i zagrożenia związane z ich użyciem

Protokół sieciowy jest standardem komunikacyjnym, który określa zasady wymiany danych pomiędzy urządzeniami w sieci. Istnieje wiele zagrożeń związanych z używaniem protokołów w sieci:

Zagrożenia związane z protokołami:

- **Słabe szyfrowanie**: Niektóre protokoły, jak HTTP, nie oferują szyfrowania danych, co umożliwia podsłuchiwanie danych przez atakujących.
- **Ataki typu Man-in-the-Middle (MITM)**: Atakujący mogą przechwycić i modyfikować dane przesyłane przez protokół, jeśli dane nie są odpowiednio zabezpieczone.
- **Brak uwierzytelniania**: Protokół może nie mieć mechanizmów uwierzytelniania, co pozwala na przejęcie połączenia przez osoby trzecie.

Przykłady zabezpieczeń:

- **SSL/TLS**: Używany do zabezpieczania połączeń internetowych, zapewnia szyfrowanie i uwierzytelnianie.
- **VPN (Virtual Private Network)**: Stosowanie VPN w celu szyfrowania połączenia między użytkownikiem a siecią.
- **SSH**: Używanie protokołu SSH do bezpiecznego zdalnego logowania, zamiast protokołu Telnet.

5. Rodzaje zagrożeń sieciowych (np. ataki DoS, spoofing, phishing)

1. **Ataki DoS (Denial of Service)**:
    - **Opis**: Celem ataku DoS jest przeciążenie systemu lub sieci, co uniemożliwia prawidłowe funkcjonowanie usług. Atakujący wysyła dużą liczbę żądań w celu wyczerpania zasobów serwera.
    - **Przykład**: Flooding – zalewanie serwera żądaniami HTTP.
2. **Spoofing**:
    - **Opis**: Atakujący podszywa się pod inne urządzenie lub użytkownika, zmieniając nagłówki wiadomości, aby wyglądały jakby pochodziły z innego źródła.
    - **Przykład**: IP spoofing – podszywanie się pod adres IP innego urządzenia.
3. **Phishing**:
    - **Opis**: Atak polegający na oszukaniu użytkownika w celu wyłudzenia danych wrażliwych, takich jak hasła, numery kart kredytowych itp.
    - **Przykład**: Fałszywe e-maile, które udają powiadomienia od banków, mające na celu wyłudzenie danych użytkownika.

6. Podstawowe mechanizmy zabezpieczające (firewall, VPN, IDS/IPS)

1. **Firewall**:
    - **Opis**: Firewall to system zabezpieczeń, który kontroluje i filtruje ruch sieciowy przychodzący i wychodzący z sieci komputerowej. Może blokować lub zezwalać na połączenia na podstawie określonych reguł.
    - **Zastosowanie**: Ochrona sieci przed nieautoryzowanym dostępem i atakami z sieci zewnętrznych.
2. **VPN (Virtual Private Network)**:

    - **Opis**: VPN tworzy bezpieczny tunel między urządzeniem użytkownika a siecią docelową, szyfrując wszystkie przesyłane dane. Jest to istotne w zapewnianiu prywatności i ochrony danych w sieci publicznej.
    - **Zastosowanie**: Umożliwia bezpieczne połączenia w sieci publicznej, takiej jak Internet.
3. **IDS/IPS (Intrusion Detection/Prevention Systems)**:
    - **Opis**: IDS to system wykrywania intruzów, który monitoruje ruch w sieci i wykrywa podejrzane działania, natomiast IPS to system zapobiegania intruzjom, który podejmuje działania w celu zatrzymania ataku.
    - **Zastosowanie**: Monitorowanie sieci pod kątem nieautoryzowanych prób dostępu oraz ataków.

1. Omów możliwości współczesnych profesjonalnych urządzeń sieciowych.
2. Omów tablicę routingu i protokoły routingu dynamicznego.
3. Omów zasadę działania VPN.

1. Omów możliwości współczesnych profesjonalnych urządzeń sieciowych

Współczesne profesjonalne urządzenia sieciowe, takie jak **routery**, **switche**, **firewalle**, **punktów dostępowe** (AP) oraz **load balancery**, oferują szereg zaawansowanych funkcji, które wspomagają zarządzanie ruchem sieciowym, poprawiają bezpieczeństwo i zapewniają dużą wydajność w sieciach komputerowych.

Główne możliwości współczesnych urządzeń sieciowych:

Router

Funkcje i zadania:

- **Routing**: Router analizuje adresy IP pakietów i decyduje, do której sieci lub urządzenia przesłać dany pakiet.
- **Filtracja pakietów**: Niektóre routery (zwłaszcza z funkcjami zapory sieciowej) mogą filtrować ruch, pozwalając lub blokując konkretne pakiety w zależności od określonych zasad.
- **NAT (Network Address Translation)**: Routery mogą zmieniać adresy IP w pakietach, co pozwala na ukrycie adresów IP w sieci wewnętrznej za jednym publicznym adresem IP.
- **DHCP**: Routery często pełnią rolę serwera DHCP, który dynamicznie przydziela adresy IP urządzeniom w sieci lokalnej.

Switch

Funkcje i zadania:

- **Przełączanie ramek**: Switch odbiera ramki danych i przekazuje je tylko do odpowiednich urządzeń, na podstawie adresu MAC urządzenia docelowego.

- **Segmentacja sieci**: Dzięki switchowi sieć może zostać podzielona na mniejsze segmenty, co zmniejsza liczbę kolizji w sieci.
- **Obsługa VLAN**: Switch może wspierać **wirtualne sieci LAN (VLAN)**, które pozwalają na logiczne podział fizycznej sieci na wiele odrębnych, izolowanych segmentów.

Firewall

Funkcje i zadania:

- **Filtrowanie pakietów**: Firewall analizuje pakiety sieciowe (np. na podstawie adresów IP, portów, protokołów) i blokuje te, które są niezgodne z ustalonymi zasadami bezpieczeństwa.
- **Kontrola aplikacji**: Zaawansowane firewalle mogą analizować dane na poziomie aplikacji, blokując np. dostęp do konkretnych aplikacji lub usług.
- **Monitorowanie ruchu**: Firewalle często monitorują całą komunikację sieciową, identyfikując podejrzany ruch, np. ataki typu DDoS.
- **VPN**: Firewall może również pełnić funkcję zarządzania bezpiecznymi połączeniami VPN.

Access Point

Funkcje i zadania:

- **Przyjmowanie i przekazywanie danych**: AP odbiera dane z urządzeń bezprzewodowych i przesyła je do routera lub switcha, który odpowiada za ich dalszą transmisję w sieci.
- **Zarządzanie ruchem**: Nowoczesne AP mogą zarządzać ruchem sieciowym, zapewniając odpowiednią jakość połączenia (QoS) i optymalizując dostępność pasma.
- **Obsługa WPA2/WPA3**: Punkty dostępowe zapewniają zabezpieczenia w sieci Wi-Fi, takie jak szyfrowanie WPA2 lub WPA3, aby zapewnić prywatność i bezpieczeństwo połączeń.

Load Balancer

Funkcje i zadania:

- **Równoważenie obciążenia**: Load balancer decyduje, który serwer ma obsłużyć dany żądanie, rozkładając ruch równomiernie, co zapobiega przeciążeniom.
- **Wysoka dostępność**: W przypadku awarii jednego z serwerów, load balancer automatycznie przekierowuje ruch na inne dostępne serwery.
- **Skalowanie**: Load balancer pozwala na dynamiczne dodawanie lub usuwanie serwerów w odpowiedzi na zmieniające się wymagania obciążeniowe.
- **Monitorowanie**: Monitoruje stan serwerów i ich wydajność, przekierowując ruch tylko do zdrowych serwerów.

1. **Routing i zarządzanie ruchem**:

- **Zaawansowane routowanie**: Urządzenia sieciowe, szczególnie routery, obsługują różnorodne protokoły routingu (np. RIP, OSPF, BGP), co pozwala na dynamiczne określanie najlepszej trasy dla danych w sieci.
- **Quality of Service (QoS)**: Umożliwia kontrolowanie priorytetów przesyłanych danych, co jest szczególnie ważne w przypadku aplikacji wymagających dużej przepustowości, np. VoIP czy transmisji wideo.
- **Dynamiczne przypisywanie tras**: Urządzenia wspierają automatyczne i elastyczne dostosowywanie tras w przypadku awarii, co zapewnia wysoką dostępność i niezawodność sieci.

2. **Zabezpieczenia i kontrola dostępu**:
   - **Firewall (zapora sieciowa)**: Nowoczesne urządzenia oferują zaawansowane funkcje zapór sieciowych, takie jak filtracja pakietów, kontrola aplikacji, zapobieganie włamaniom (IDS/IPS).
   - **VPN (Virtual Private Network)**: Umożliwiają bezpieczne połączenia między oddzielnymi sieciami lub zdalnymi użytkownikami, szyfrując dane i zapewniając ochronę przed podsłuchiwaniem.
   - **Uwierzytelnianie i autoryzacja**: Obsługuje protokoły takie jak RADIUS, TACACS+, które pozwalają na centralne zarządzanie dostępem użytkowników do zasobów sieciowych.
3. **Zarządzanie siecią**:
   - **SNMP (Simple Network Management Protocol)**: Protokół umożliwiający monitorowanie i zarządzanie urządzeniami sieciowymi z centralnego punktu, co pozwala na szybkie reagowanie na problemy.
   - **Zarządzanie przez interfejs webowy i CLI**: Większość profesjonalnych urządzeń oferuje możliwość konfiguracji i monitorowania przez prosty interfejs webowy oraz zaawansowane opcje konfiguracji w trybie wiersza poleceń.
4. **Wydajność i skalowalność**:
   - **Wielkie przepustowości**: Urządzenia sieciowe nowej generacji oferują obsługę dużych przepustowości w sieciach 10 GbE, 40 GbE, 100 GbE i wyższych.
   - **Redundancja i wysoką dostępność**: Wbudowane mechanizmy zapewniające redundancję w postaci protokołów takich jak HSRP, VRRP, które zapewniają ciągłość działania sieci w razie awarii urządzeń.

2. Omów tablicę routingu i protokoły routingu dynamicznego

Tablica routingu:

Tablica routingu to struktura danych przechowywana w każdym routerze, która zawiera informacje o dostępnych trasach sieciowych. Routery wykorzystują tę tabelę, aby dowiedzieć się, jak przesłać pakiety do ich docelowych adresów. Tablica routingu może zawierać:

- **Adresy docelowe**: Określa sieci, do których router może przesyłać pakiety.
- **Maski podsieci**: Określają, które bity w adresie IP wskazują na część sieciową.
- **Kroki (hopy)**: Liczba przeskoków, jakie musi pokonać pakiet, aby dotrzeć do celu.
- **Interfejsy wyjściowe**: Określenie, przez który interfejs routera pakiet powinien zostać wysłany.
- **Protokół routingu**: Źródło informacji o trasie (np. RIP, OSPF, BGP).

Tablica routingu może być statyczna (ręcznie skonfigurowana przez administratora) lub dynamiczna (aktualizowana przez protokoły routingu dynamicznego).

Protokoły routingu dynamicznego:

Protokoły routingu dynamicznego umożliwiają routerom wymianę informacji o trasach i automatyczne aktualizowanie tablic routingu na podstawie zmieniającej się topologii sieci.

1. **RIP (Routing Information Protocol)**:
    - **Typ**: Protokół wektora odległości.
    - **Zasada działania**: Routery wymieniają się informacjami o trasach i obliczają najkrótszą trasę na podstawie liczby przeskoków (maksymalnie 15).
    - **Zalety**: Prosty, łatwy do skonfigurowania.
    - **Wady**: Ograniczenie liczby przeskoków (maksymalnie 15), nieefektywny w większych sieciach.
2. **OSPF (Open Shortest Path First)**:
    - **Typ**: Protokół link-state.
    - **Zasada działania**: Każdy router w sieci buduje mapę topologii sieci, a na podstawie tej mapy oblicza najkrótsze ścieżki do innych routerów za pomocą algorytmu Dijkstry.
    - **Zalety**: Skalowalność, szybka reakcja na zmiany topologii.
    - **Wady**: Wymaga więcej zasobów do przechowywania i obliczeń.
3. **BGP (Border Gateway Protocol)**:
    - **Typ**: Protokół wektora ścieżki.
    - **Zasada działania**: BGP wymienia informacje o trasach pomiędzy różnymi systemami autonomicznymi (AS) w Internecie, uwzględniając wiele czynników przy wyborze trasy.
    - **Zalety**: Wysoka skalowalność i elastyczność, kluczowy protokół w Internecie.
    - **Wady**: Złożoność konfiguracji, wrażliwość na ataki, takie jak prefix hijacking.

3. Omów zasadę działania VPN

VPN (Virtual Private Network) to technologia, która pozwala na tworzenie bezpiecznych i zaszyfrowanych połączeń pomiędzy użytkownikiem a siecią, nawet jeśli użytkownik korzysta z publicznych sieci, jak np. Internet. VPN jest wykorzystywany do zapewnienia prywatności i bezpieczeństwa podczas przesyłania danych.

Zasada działania VPN:

1. **Szyfrowanie**:
    - VPN szyfruje dane przesyłane między urządzeniem użytkownika a serwerem VPN, co zapewnia, że nawet jeśli dane będą przechwycone, nie będą mogły zostać odczytane przez osoby trzecie.
2. **Tunelowanie**:
    - VPN tworzy „tunel” między urządzeniem użytkownika a serwerem VPN. Dzięki temu cały ruch przechodzący przez tunel jest szyfrowany, co zapobiega jego podsłuchiwaniu przez osoby postronne.
3. **Protokół VPN**:
    - Istnieje kilka protokołów VPN, które określają sposób tworzenia i utrzymywania bezpiecznego połączenia:
        - **PPTP (Point-to-Point Tunneling Protocol)**: Stary protokół, szybki, ale mniej bezpieczny.
        - **L2TP/IPsec (Layer 2 Tunneling Protocol)**: Łączy tunelowanie L2TP z szyfrowaniem IPsec, zapewniając lepsze bezpieczeństwo.

        - **OpenVPN**: Otwarty protokół VPN, uważany za jeden z najbezpieczniejszych.
        - **IKEv2/IPsec**: Protokół VPN oferujący wysoką wydajność i bezpieczeństwo, odporny na zmiany sieci (np. przełączanie między Wi-Fi a 4G).
4. **Autentykacja i weryfikacja tożsamości**:
    - VPN zapewnia uwierzytelnianie użytkownika i serwera, aby upewnić się, że obie strony połączenia są tymi, za które się podają. Uwierzytelnianie może być oparte na haśle, certyfikatach cyfrowych lub innych mechanizmach.
5. **Korzyści z używania VPN**:
    - **Bezpieczeństwo**: Szyfrowanie danych zapewnia, że są one chronione przed podsłuchiwaniem.
    - **Prywatność**: VPN ukrywa adres IP użytkownika, co utrudnia jego identyfikację i śledzenie w Internecie.
    - **Zdalny dostęp**: Umożliwia bezpieczne łączenie się z zasobami firmy lub siecią prywatną z dowolnego miejsca na świecie.
    - **Ochrona przed atakami**: VPN może chronić użytkowników przed atakami, takimi jak Man-in-the-Middle (MITM), w których atakujący przechwytują dane w sieci.

Programowanie obiektowe 2

1. Własne klasy, tworzenie instancji obiektów, pojęcie konstruktora/inicjalizatora, metody i atrybuty instancyjne. Atrybuty i metody klasowe

- **Własne klasy**: Klasa w Pythonie to szablon do tworzenia obiektów. Klasa definiuje atrybuty (dane) i metody (funkcje), które są dostępne dla jej obiektów.

```
class Person:
  def __init__(self, name, age):
    self.name = name  # Atrybut instancyjny
    self.age = age    # Atrybut instancyjny

  def greet(self):  # Metoda instancyjna
    print(f"Hello, my name is {self.name} and I'm {self.age} years old.")
```

- **Tworzenie instancji obiektów**: Instancje obiektów są tworzone przez wywołanie klasy jak funkcji.

```
person1 = Person("John", 30)
person1.greet()  # Wywołanie metody instancyjnej
```

- **Konstruktor/Inicjalizator (__init__)**: To specjalna metoda, która jest wywoływana przy tworzeniu instancji obiektu. Służy do inicjalizacji atrybutów instancyjnych.

```
def __init__(self, name, age):
  self.name = name
  self.age = age
```

- **Atrybuty i metody instancyjne**: Atrybuty instancyjne to dane przypisane do obiektu, a metody instancyjne to funkcje, które mogą operować na tych danych.

- **Atrybuty i metody klasowe**: Atrybuty klasowe są współdzielone przez wszystkie instancje klasy. Metody klasowe (oznaczone dekoratorem @classmethod) operują na klasie, nie na jej instancjach.

```
class MyClass:
    class_var = 0  # Atrybut klasowy

    def __init__(self, value):
        self.instance_var = value  # Atrybut instancyjny

    @classmethod
    def class_method(cls):
        print(f"Class var: {cls.class_var}")  # Odwołanie do atrybutu klasowego
```

2. Pojęcie obiektu w języku Python. Sposoby reprezentacji obiektów w języku Python. Pojęcie obiektu iterowalnego i iteratora.

- **Obiekt w Pythonie**: Obiekt to instancja klasy, która posiada swoje dane (atrybuty) i może wykonywać operacje (metody). Każdy obiekt w Pythonie jest przechowywany w pamięci jako instancja klasy.

```
class Car:
    def __init__(self, brand, model):
        self.brand = brand
        self.model = model

car = Car("Toyota", "Corolla")  # car jest obiektem klasy Car
```

- **Reprezentacja obiektów w Pythonie**: Obiekty w Pythonie są przechowywane w pamięci jako instancje klasy, a każdy obiekt posiada unikalny identyfikator (ID). Reprezentacja obiektu jako tekst jest zdefiniowana w metodzie __repr__ lub __str__.

```
def __repr__(self):
    return f"Car(brand='{self.brand}', model='{self.model}')"
```

- **Obiekt iterowalny**: Obiekt iterowalny to taki, który może być używany w pętli for. Obiekty te implementują metodę __iter__() i zwracają obiekt iteratora.

```
class MyRange:
    def __init__(self, start, end):
        self.start = start
        self.end = end

    def __iter__(self):
        self.current = self.start
        return self

    def __next__(self):
        if self.current < self.end:
            self.current += 1
            return self.current - 1
        else:
            raise StopIteration
```

- **Iterator**: Iterator to obiekt, który przechowuje stan iteracji i zwraca elementy obiektu iterowalnego, kiedy wywołamy funkcję next().

3. Właściwości - zastosowania i sposób tworzenia. Przeciążenie operatorów.

- **Właściwości (properties)**: Właściwości umożliwiają kontrolowanie dostępu do atrybutów obiektów. Można je definiować za pomocą dekoratorów @property i @setter.

```
class Circle:
    def __init__(self, radius):
        self._radius = radius

    @property
    def radius(self):
        return self._radius

    @radius.setter
    def radius(self, value):
        if value < 0:
            raise ValueError("Radius cannot be negative")
        self._radius = value
```

- **Przeciążenie operatorów**: możemy przeciążyć operatory, definiując specjalne metody w klasie, np. __add__ dla operatora +, __eq__ dla operatora ==.

```
class Point:
    def __init__(self, x, y):
        self.x = x
        self.y = y

    def __add__(self, other):
        return Point(self.x + other.x, self.y + other.y)
```

4. Dziedziczenie i agregacja/kompozycja w języku Python. Działanie funkcji super(). Protokoły, interfejsy i abstrakcyjne klasy bazowe w języku Python.

- **Dziedziczenie**: Dziedziczenie pozwala na tworzenie nowych klas na podstawie już istniejących. Klasa potomna dziedziczy atrybuty i metody klasy bazowej.

```
class Animal:
    def speak(self):
        print("Animal speaks")

class Dog(Animal):
    def speak(self):
        print("Woof!")
```

- **Agregacja/Kompozycja**: Agregacja to stosunek "ma" (np. samochód posiada silnik). Kompozycja to silniejszy związek, gdzie obiekt nie może istnieć bez obiektu zawierającego go (np. "dom" posiada "pokój").

- **Funkcja `super()`**: Używana do wywołania metod klasy bazowej. Jest szczególnie przydatna w metodach nadpisanych.

```
class Dog(Animal):
    def speak(self):
        super().speak()  # Wywołanie metody klasy bazowej
        print("Woof!")
```

- **Protokoły, interfejsy i abstrakcyjne klasy bazowe**:
    - **Protokoły**: Określają, jak obiekt ma się zachowywać, nie definiując szczegółowej implementacji (zastosowanie w programowaniu funkcyjnym).
    - **Interfejsy**: W Pythonie nie ma interfejsów jak w innych językach, ale można używać klas abstrakcyjnych lub protokołów z modułu `typing`.
    - **Abstrakcyjne klasy bazowe (ABC)**: To klasy, które mogą zawierać metody abstrakcyjne (niezaimplementowane), które muszą zostać zaimplementowane w klasach potomnych.

```
from abc import ABC, abstractmethod

class Animal(ABC):
    @abstractmethod
    def speak(self):
        pass
```

5. Python jako język wspierający paradygmat funkcyjny

Python wspiera paradygmat funkcyjny, co oznacza, że funkcje są obywatelami pierwszej klasy. Można je przypisywać do zmiennych, przekazywać jako argumenty i zwracać z innych funkcji. Python oferuje także funkcje wyższego rzędu (np. `map()`, `filter()`), funkcje anonimowe (lambda) oraz obsługę funkcji rekurencyjnych.

```
# Funkcja wyższego rzędu
def apply_func(f, value):
    return f(value)

print(apply_func(lambda x: x + 1, 10))
```

6. Deskryptory, ich rodzaje i zastosowania w języku Python oraz pojęcie metaklas

- **Deskryptory**: Są to obiekty, które kontrolują dostęp do atrybutów obiektów. Implementują co najmniej jedną z metod: `__get__`, `__set__`, `__delete__`.

```
class Descriptor:
    def __get__(self, instance, owner):
        return "Accessing value"
```

- **Metaklasy**: Metaklasa to klasa, która tworzy klasy. Metaklasy umożliwiają kontrolowanie tworzenia klas, np. ich struktur, metod czy atrybutów.

```
class MyMeta(type):
```

```
    def __new__(cls, name, bases, dct):
        print(f"Creating class {name}")
        return super().__new__(cls, name, bases, dct)

class MyClass(metaclass=MyMeta):
    pass
```

Metaklasy są używane do modyfikowania sposobu, w jaki klasy są tworzone w Pythonie, pozwalając na dodanie dodatkowych mechanizmów, jak walidacja atrybutów lub metod.

1. Małpie łatanie (Monkey Patching)

Małpie łatanie (ang. **Monkey Patching**) to technika programistyczna, w której modyfikuje się lub rozszerza działanie istniejących klas lub modułów w trakcie działania programu, czyli "w locie". Zmiana jest wprowadzana bezpośrednio do kodu obiektów lub metod, co może być niebezpieczne, ale daje dużą elastyczność.

W Pythonie, "małpie łatanie" najczęściej polega na nadpisaniu istniejącej metody w klasie lub funkcji bez zmiany jej definicji w oryginalnym kodzie.

Przykład:

Załóżmy, że mamy klasę `Car`, a chcemy, żeby każda instancja tej klasy miała dodatkową metodę `start_engine`.

```
class Car:
    def start_engine(self):
        print("Starting engine...")

# Monkey patching: dodajemy metodę 'stop_engine' do klasy Car w czasie działania programu
def stop_engine(self):
    print("Stopping engine...")

Car.stop_engine = stop_engine

# Teraz możemy wywołać stop_engine na instancji klasy Car
car = Car()
car.start_engine()  # "Starting engine..."
car.stop_engine()   # "Stopping engine..."
```

Zalety:

- Elastyczność: Pozwala na szybkie poprawki w kodzie zewnętrznych bibliotek, których nie możemy modyfikować bezpośrednio.
- Testowanie: Można tymczasowo zmieniać zachowanie klas lub funkcji, aby testować różne przypadki.

Wady:

- Trudności w utrzymaniu: Może prowadzić do trudnych do wykrycia błędów.
- Nieczytelność: Używanie tej techniki może sprawić, że kod stanie się mniej przejrzysty, zwłaszcza w dużych projektach.

2. Kacze Typowanie (Duck Typing)

Kacze typowanie (ang. **Duck Typing**) to podejście w programowaniu, które jest stosowane głównie w językach dynamicznych, takich jak Python. Termin ten pochodzi od przysłowia: "Jeśli coś chodzi jak kaczka i kwacze jak kaczka, to zapewne jest kaczką". Oznacza to, że nie ma potrzeby sprawdzania, czy obiekt należy do konkretnego typu (np. klasy), wystarczy sprawdzić, czy obiekt ma odpowiednie właściwości lub metody.

W Pythonie oznacza to, że nie musimy jawnie deklarować typów zmiennych, ponieważ jeśli obiekt obsługuje wymagane metody i atrybuty, to traktujemy go jako odpowiedni typ.

Przykład:

```
class Bird:
    def speak(self):
        print("Chirp chirp!")

class Dog:
    def speak(self):
        print("Woof woof!")

def make_speak(animal):
    animal.speak()  # Zauważ, że nie sprawdzamy typu obiektu!

bird = Bird()
dog = Dog()

make_speak(bird)  # "Chirp chirp!"
make_speak(dog)   # "Woof woof!"
```

W powyższym przykładzie zarówno klasa `Bird`, jak i `Dog` mają metodę `speak()`. Dzięki kaczymu typowaniu możemy przekazać oba obiekty do funkcji `make_speak()` bez konieczności sprawdzania, czy są one instancjami jakiejś klasy bazowej.

Zalety:

- Elastyczność: Kiedy obiekt ma odpowiednie metody, może być używany w dowolnym kontekście.
- Przejrzystość: Kod jest czystszy, ponieważ nie ma potrzeby deklarowania typów.

Wady:

- Brak kontroli typów: W błędnych przypadkach brak jawnego sprawdzania typów może prowadzić do błędów, które pojawią się dopiero w czasie działania programu (np. jeśli obiekt nie ma metody `speak()`).

3. Inne podobne pojęcia

- **Polimorfizm**: W kontekście kacze typowanie, polimorfizm odnosi się do zdolności obiektów różnych typów do korzystania z tych samych metod, jeśli implementują te same interfejsy, zachowując przy tym różne zachowanie. Kacze typowanie jest jednym z aspektów polimorfizmu w językach dynamicznych.
- **Typowanie dynamiczne**: Jest to cecha języków takich jak Python, w którym typ zmiennej jest określany w czasie działania programu, a nie na etapie kompilacji. Pozwala to na elastyczność w używaniu obiektów, ale także wiąże się z ryzykiem błędów związanych z nieodpowiednim typem.

Bezpieczeństwo Informacji

1. Podstawowe elementy modelu CIA (poufność, integralność, dostępność)

Model CIA (Confidentiality, Integrity, Availability) to fundamenty zapewnienia bezpieczeństwa informacji w systemach komputerowych i sieciach. Opisuje on trzy kluczowe aspekty ochrony danych:

- **Poufność (Confidentiality)**: Gwarantuje, że dostęp do danych mają tylko uprawnione osoby lub systemy. Oznacza to ochronę danych przed nieautoryzowanym dostępem. Poufność jest zapewniana przez metody szyfrowania, kontrolę dostępu i polityki prywatności.
- **Integralność (Integrity)**: Dotyczy zapewnienia, że dane nie zostały zmienione w sposób nieautoryzowany, zarówno w trakcie przechowywania, jak i podczas transmisji. Integralność danych można zapewnić za pomocą sum kontrolnych (hashy), podpisów cyfrowych oraz mechanizmów wykrywania i naprawy błędów.
- **Dostępność (Availability)**: Zapewnia, że dane i usługi są dostępne i działają poprawnie, gdy są potrzebne. Dostępność jest kluczowa, zwłaszcza w systemach krytycznych. Mechanizmy zapewniające dostępność obejmują redundancję, backupy, ochronę przed atakami DoS (Denial of Service) i procedury odzyskiwania po awarii.

2. Podstawy szyfrowania: algorytmy symetryczne i asymetryczne

Szyfrowanie jest procesem, który przekształca dane w formę, która jest trudna do odczytania przez osoby nieuprawnione. Istnieją dwa główne typy algorytmów szyfrowania:

- **Algorytmy symetryczne**: W tym przypadku zarówno proces szyfrowania, jak i deszyfrowania wykorzystują ten sam klucz. Klucz musi być utrzymywany w tajemnicy między nadawcą a odbiorcą. Przykłady algorytmów symetrycznych to:
    - **AES (Advanced Encryption Standard)**
    - **DES (Data Encryption Standard)**
    - **3DES (Triple DES)**

  Zalety: Szybkość, wydajność, łatwość implementacji.

  Wady: Problem dystrybucji kluczy, ponieważ klucz musi być wymieniany w sposób bezpieczny.

- **Algorytmy asymetryczne**: Używają dwóch kluczy: publicznego i prywatnego. Klucz publiczny jest dostępny dla każdego, natomiast klucz prywatny jest trzymany w tajemnicy przez właściciela. Szyfrowanie odbywa się za pomocą klucza publicznego, a deszyfrowanie za pomocą klucza prywatnego. Przykłady algorytmów asymetrycznych to:
    - **RSA**
    - **ECC (Elliptic Curve Cryptography)**
    - **DSA (Digital Signature Algorithm)**

  Zalety: Umożliwia bezpieczne przesyłanie kluczy, nie wymaga wcześniejszego udostępniania klucza prywatnego.

  Wady: Wolniejsza wydajność w porównaniu do szyfrowania symetrycznego.

3. Podpis elektroniczny i infrastruktura klucza publicznego (PKI)

- **Podpis elektroniczny**: Jest to mechanizm, który umożliwia uwierzytelnienie autora dokumentu i zapewnienie integralności danych. Podpisy elektroniczne działają na zasadzie algorytmów asymetrycznych – dokument jest haszowany, a wynikowy skrót jest szyfrowany kluczem prywatnym nadawcy. Odbiorca może zweryfikować podpis, używając klucza publicznego nadawcy.
- **Infrastruktura klucza publicznego (PKI)**: To system, który umożliwia bezpieczne zarządzanie kluczami publicznymi i prywatnymi oraz ich wymianę. PKI zapewnia procedury generowania, przechowywania, dystrybucji oraz unieważniania certyfikatów cyfrowych. Certyfikaty są wydawane przez zaufane jednostki zwane **Centrami Certyfikacji (CA)**, które są odpowiedzialne za autentyczność i wiarygodność kluczy publicznych.

4. Metody uwierzytelniania

Uwierzytelnianie to proces weryfikacji tożsamości użytkownika lub systemu. Istnieje kilka głównych metod uwierzytelniania:

- **Hasło**: Najczęściej stosowana metoda, w której użytkownik podaje tajne hasło. Jest to metoda słaba, jeśli hasła są łatwe do odgadnięcia lub skradzione.
- **Uwierzytelnianie dwuskładnikowe (2FA)**: Wymaga dwóch różnych form uwierzytelniania, np. hasła oraz kodu wysłanego SMS-em lub aplikacją mobilną. Zwiększa bezpieczeństwo, eliminując ryzyko kradzieży hasła.
- **Biometria**: Uwierzytelnianie za pomocą cech biologicznych użytkownika, takich jak odciski palców, skanowanie siatkówki oka, czy rozpoznawanie twarzy.
- **Uwierzytelnianie oparte na certyfikatach**: Certyfikaty cyfrowe są używane do weryfikacji tożsamości użytkowników w systemach IT.

5. Podstawy bezpieczeństwa fizycznego i logicznego

- **Bezpieczeństwo fizyczne**: Obejmuje ochronę urządzeń i danych przed kradzieżą, zniszczeniem lub nieautoryzowanym dostępem do sprzętu. Obejmuje to takie środki

jak kontrola dostępu do budynków, monitoring, sejfy, czy urządzenia antywłamaniowe.
- **Bezpieczeństwo logiczne**: Obejmuje ochronę przed nieautoryzowanym dostępem do systemów komputerowych i danych. Do podstawowych metod należy stosowanie haseł, szyfrowania, zapór sieciowych (firewalli) i programów antywirusowych.

6. Główne rodzaje cyberataków i metody obrony przed nimi

- **Ataki DDoS (Distributed Denial of Service)**: Atak polegający na przeciążeniu systemu (serwera, sieci) ogromną ilością niepotrzebnych żądań, co uniemożliwia normalne funkcjonowanie. Ochrona przed atakami DDoS może obejmować użycie zapór sieciowych, rozdzielanie ruchu na wiele serwerów, czy stosowanie systemów wykrywania i zapobiegania włamaniom (IDS/IPS).
- **Phishing**: Przestępcy podszywają się pod zaufane instytucje, aby wyłudzić dane użytkowników, takie jak hasła lub dane kart kredytowych. Ochrona przed phishingiem obejmuje edukację użytkowników, stosowanie weryfikacji dwuetapowej oraz technologie ochrony przed oszustwami (np. filtrowanie e-maili).
- **Spoofing**: Fałszowanie tożsamości w celu przejęcia kontroli nad systemem lub oszukania użytkownika. Można się przed nim chronić za pomocą uwierzytelniania, certyfikatów cyfrowych oraz mechanizmów weryfikacji tożsamości.
- **Ransomware**: Złośliwe oprogramowanie, które blokuje dostęp do danych i żąda okupu za ich odblokowanie. Ochrona przed ransomware polega na regularnym tworzeniu kopii zapasowych, stosowaniu filtrów antywirusowych oraz edukacji użytkowników o zagrożeniach.

Metody obrony:

- **Firewall**: Urządzenie lub oprogramowanie filtrujące ruch sieciowy, zapobiegające nieautoryzowanemu dostępowi do sieci.
- **VPN**: Virtual Private Network, szyfruje transmisje i zapewnia bezpieczny dostęp do zasobów sieciowych.
- **IDS/IPS**: Systemy wykrywania (Intrusion Detection System) i zapobiegania włamaniom (Intrusion Prevention System), monitorujące i analizujące ruch w sieci w celu wykrywania ataków.
- **Szyfrowanie danych**: Zapewnia poufność informacji i chroni je przed kradzieżą.

By Hubertus Bubertus