A **programming language** is a formal set of rules and symbols used to communicate instructions to a computer. It enables developers to write programs that can perform a wide variety of tasks, from simple calculations to complex algorithms and systems.

**Types of Programming Languages**

1. **Low-Level Languages**:
   * **Machine Language**: The most basic language that a computer can understand, consisting of binary code (0s and 1s).
   * **Assembly Language**: A step above machine language, it uses symbolic representations of machine code instructions. It is closely related to the architecture of the computer.
2. **High-Level Languages**:
   * **Procedural Languages**: Focus on a sequence of procedures or routines (e.g., C, Pascal).
   * **Object-Oriented Languages**: Use objects to represent data and methods (e.g., Java, C++, Python). They facilitate code reuse and modularity.
   * **Functional Languages**: Treat computation as the evaluation of mathematical functions, avoiding changing-state and mutable data (e.g., Haskell, Lisp).
   * **Scripting Languages**: Often used for automating tasks or manipulating data, they tend to be interpreted rather than compiled (e.g., JavaScript, Python, Ruby).
   * **Domain-Specific Languages**: Tailored for specific application domains (e.g., SQL for databases, HTML for web pages).
3. **Mark-Up Languages**:
   * Not programming languages in the strict sense, but they define data structures (e.g., HTML, XML). They describe the format and presentation of information rather than performing calculations.
4. **Declarative Languages**:
   * Focus on what the program should accomplish without specifying how to achieve it (e.g., SQL, Prolog).
5. **Logic Programming Languages**:
   * Use formal logic to express facts and rules about some problem domain (e.g., Prolog).

**Summary**

Each type of programming language serves different purposes and is suited to different tasks. The choice of language can depend on factors like the specific application requirements, performance needs, developer preferences, and the available libraries or frameworks.

4o mini

**You said:**

IDE

**ChatGPT said:**

ChatGPT

An **IDE** (Integrated Development Environment) is a software application that provides comprehensive facilities to programmers for software development. It typically includes a combination of the following components:

**Key Features of an IDE**

1. **Code Editor**:
   * A powerful text editor that supports syntax highlighting, code completion, and other features that make coding easier and more efficient.
2. **Compiler/Interpreter**:
   * Tools to convert the written code into machine code or to execute it directly. This allows for the testing and running of programs from within the IDE.
3. **Debugger**:
   * A built-in tool for detecting and fixing errors in the code. It allows developers to set breakpoints, step through code, and inspect variables.
4. **Build Automation Tools**:
   * Features that automate the process of compiling and linking code, making it easier to manage large projects.
5. **Version Control Integration**:
   * Tools to manage changes to the codebase using systems like Git, allowing for better collaboration among developers.
6. **User Interface Designer**:
   * Some IDEs provide visual design tools for creating user interfaces (UIs) for applications, allowing drag-and-drop placement of UI components.
7. **Project Management**:
   * Features to manage files, resources, and dependencies within a project, often displaying the project structure in a hierarchical view.
8. **Plugins and Extensions**:
   * Many IDEs support additional plugins or extensions to enhance functionality, enabling support for more languages, frameworks, and tools.

**Popular IDEs**

1. **Visual Studio**: A powerful IDE from Microsoft, widely used for .NET and C++ development.
2. **Eclipse**: An open-source IDE mainly used for Java development, but also supports other languages through plugins.
3. **IntelliJ IDEA**: A popular IDE for Java development, known for its intelligent code assistance features.
4. **PyCharm**: Specifically designed for Python development, offering a rich set of tools for web and data science projects.
5. **NetBeans**: An open-source IDE that supports multiple languages, primarily Java.
6. **Xcode**: Apple's IDE for macOS and iOS development, primarily used for Swift and Objective-C.
7. **Visual Studio Code**: A lightweight but powerful code editor that can function as an IDE with the help of extensions, supporting multiple languages.

**Summary**

IDEs significantly enhance productivity by providing tools that streamline the development process. By integrating various functionalities into a single environment, they help developers write, test, and debug code more efficiently.

4o mini

**You said:**

interpreted language

**ChatGPT said:**

ChatGPT

An **interpreted language** is a type of programming language in which most of the instructions are executed directly, line by line, by an interpreter at runtime, rather than being compiled into machine code beforehand. This contrasts with compiled languages, where the entire program is translated into machine code before execution.

**Characteristics of Interpreted Languages**

1. **Line-by-Line Execution**: The interpreter reads and executes the code one line at a time. This can make debugging easier, as errors can be caught immediately when a specific line is executed.
2. **No Separate Compilation Step**: Code can be executed without a separate compilation step, making the development process faster and more flexible.
3. **Portability**: Since interpreted languages are executed by an interpreter, the same code can often run on any platform that has the appropriate interpreter.
4. **Dynamic Typing**: Many interpreted languages are dynamically typed, allowing for more flexible variable handling. However, this can lead to runtime errors that would be caught at compile-time in statically typed languages.
5. **Ease of Use**: Interpreted languages are often considered easier to learn and use, particularly for beginners, due to their straightforward syntax and immediate feedback during execution.

**Examples of Interpreted Languages**

1. **Python**: Widely used for web development, data analysis, artificial intelligence, and more.
2. **JavaScript**: Primarily used for web development, running in browsers to create dynamic web applications.
3. **Ruby**: Known for its elegant syntax, often used in web development (e.g., Ruby on Rails).
4. **PHP**: A server-side scripting language designed primarily for web development.
5. **Perl**: Often used for text processing and system administration tasks.

**Advantages of Interpreted Languages**

* **Rapid Development**: Changes can be tested immediately without a compilation step, speeding up the development process.
* **Flexibility**: Easier to write code that can adapt to different data types or structures at runtime.
* **Interactive Execution**: Many interpreted languages support interactive modes, allowing for immediate execution of commands.

**Disadvantages of Interpreted Languages**

* **Performance**: Interpreted languages tend to be slower than compiled languages because the interpretation happens at runtime.
* **Runtime Errors**: Errors may only be discovered when a specific piece of code is executed, which can lead to issues if not properly tested.

**Summary**

Interpreted languages are powerful tools for rapid application development and are widely used in various domains, especially web and scripting environments. They offer advantages in ease of use and flexibility, making them popular choices for many developers.

**Dynamically Typed** and **Statically Typed** are terms that describe how a programming language handles variable types and type checking.

**Dynamically Typed Languages**

In dynamically typed languages, variable types are determined at runtime, meaning you don’t need to declare the type of a variable when you create it. The interpreter checks the variable types as the program runs.

**Characteristics:**

1. **Type Inference**: The type of a variable can change during execution.
2. **Ease of Use**: Often simpler and more flexible to work with, especially for beginners.
3. **Less Verbose**: Typically requires fewer lines of code, as type declarations are not necessary.

Top of Form

**Advantages:**

* **Flexibility**: You can easily change types without modifying the code significantly.
* **Rapid Development**: Faster prototyping and development due to less boilerplate code.

**Disadvantages:**

* **Runtime Errors**: Type-related errors may only be caught during execution, making debugging more challenging.
* **Performance**: Can be slower than statically typed languages, as type checking happens at runtime.

**Statically Typed Languages**

In statically typed languages, variable types are determined at compile-time. You must declare the type of a variable before using it, and the compiler checks types during the compilation process.

**Characteristics:**

1. **Type Declaration**: Variables must be declared with a specific type, and this type cannot change.
2. **Early Error Detection**: Type-related errors are caught during compilation, reducing runtime errors.
3. **Improved Performance**: Often more efficient since types are known at compile time.

**Advantages:**

1. **Type Safety**: Errors are detected early in the development process.
2. **Better Performance**: Generally more efficient execution due to compile-time optimizations.
3. **Disadvantages:**
4. **Verbosity**: Requires more code to declare types, which can slow down development.
5. **Less Flexibility**: Changing the type of a variable requires modifying the type declaration.
6. **Summary**
7. **Dynamically Typed**: Types are checked at runtime. More flexible, easier to write, but can lead to runtime errors.
8. **Statically Typed**: Types are checked at compile time. More type-safe, potentially better performance, but requires more upfront type management.
9. The choice between dynamic and static typing often depends on the specific requirements of a project, including considerations for performance, safety, and ease of use.
10. 4o mini

Bottom of Form

**1. What is Python? What are the benefits of using Python Python is a high-level, interpreted, general-purpose programming language. Being a general-purpose language, it can be used to build almost any type of application with the right tools/libraries. Additionally, python supports objects, modules, threads, exception-handling, and automatic memory management which help in modelling real-world problems and building applications to solve these problems. Page 7 © Copyright by Interviewbit Benefits of using Python: Python Interview Questions Python is a general-purpose programming language that has a simple, easy-to learn syntax that emphasizes readability and therefore reduces the cost of program maintenance. Moreover, the language is capable of scripting, is completely open-source, and supports third-party packages encouraging modularity and code reuse. Its high-level data structures, combined with dynamic typing and dynamic binding, attract a huge community of Development and deployment.**

Benefits of using Python: Python is a general-purpose programming language that has a simple, easy-to learn syntax that emphasizes readability and therefore reduces the cost of program maintenance. Moreover, the language is capable of scripting, is completely open-source, and supports third-party packages encouraging modularity and code reuse. Its high-level data structures, combined with dynamic typing and dynamic binding, attract a huge community of Development and deployment.

In Python, **modules** and **packages** are important concepts that help organize and structure code for better maintainability and reusability. Here’s an overview of each:

**Python Modules**

A **module** is a single file (with a .py extension) that contains Python code. This code can define functions, classes, and variables. Modules allow you to logically organize your Python code into manageable sections.

**Key Features:**

* **Reusability**: You can import a module into other Python scripts to use its functions and classes, promoting code reuse.
* **Namespace**: Modules provide a separate namespace, meaning that functions or variables defined in one module do not conflict with those in another.

**Lambda Functions**

**Lambda functions** in Python are small, anonymous functions defined using the lambda keyword. They can take any number of arguments but can only have a single expression. The result of the expression is returned automatically.

**Key Features:**

* **Syntax**: lambda arguments: expression
* **Anonymous**: Unlike regular functions defined with def, lambda functions do not have a name.
* **Single Expression**: They are limited to a single expression, making them less powerful than regular functions but useful for small operations.

**kwargs (Keyword Arguments)**

**kwargs** stands for "keyword arguments." It allows you to pass a variable number of keyword arguments (i.e., named arguments) to a function. The kwargs parameter is defined with two asterisks (\*\*) before its name in the function definition.

**Key Features:**

* **Flexibility**: It allows functions to accept an arbitrary number of keyword arguments.
* **Dictionary**: Inside the function, kwargs is treated as a dictionary where the keys are the argument names and the values are the corresponding argument values.
* **Combining Lambda and kwargs**
* You can combine lambda functions and kwargs to create concise and flexible code. For example, you can use a lambda function as a value in a dictionary that is passed to a function via kwargs.

Summary

Lambda Functions: Small, anonymous functions useful for short, throwaway functions. They are defined using the lambda keyword and can take any number of arguments but only a single expression.

kwargs: A way to pass a variable number of keyword arguments to a function. Inside the function, they are accessed as a dictionary.

**PYTHON DECORATORS**

Python decorators are a powerful and expressive way to modify or enhance the behavior of functions or methods. They allow you to "wrap" a function with another function, adding functionality before or after the original function is executed, without changing the original function's code.

**Basic Syntax**

A decorator is typically defined as a function that takes another function as an argument and returns a new function. Here's a simple example:

python

Copy code

def my\_decorator(func):

def wrapper():

print("Something is happening before the function is called.")

func()

print("Something is happening after the function is called.")

return wrapper

@my\_decorator

def say\_hello():

print("Hello!")

say\_hello()

**Output**

vbnet

Copy code

Something is happening before the function is called.

Hello!

Something is happening after the function is called.

**How it Works**

1. **Function Definition**: The my\_decorator function is defined to take a function (func) as its argument.
2. **Wrapper Function**: Inside my\_decorator, a wrapper function is defined, which adds behavior before and after calling the original function (func).
3. **Decorator Syntax**: The @my\_decorator syntax above the say\_hello function applies the decorator. It's equivalent to say\_hello = my\_decorator(say\_hello).

**Decorators with Arguments**

If you need your decorator to accept arguments, you can define it as a function that returns another decorator:

python

Copy code

def repeat(num\_times):

def decorator\_repeat(func):

def wrapper(\*args, \*\*kwargs):

for \_ in range(num\_times):

func(\*args, \*\*kwargs)

return wrapper

return decorator\_repeat

@repeat(num\_times=3)

def greet(name):

print(f"Hello, {name}!")

greet("Alice")

**Output**

Copy code

Hello, Alice!

Hello, Alice!

Hello, Alice!

**Common Use Cases**

1. **Logging**: Automatically log function calls and their arguments.
2. **Authorization**: Check if a user has permission to execute a function.
3. **Memoization**: Cache the results of expensive function calls to optimize performance.
4. **Timing**: Measure the execution time of a function.

**Built-in Decorators**

Python also includes several built-in decorators like:

* @staticmethod: Defines a method that doesn't operate on an instance of the class.
* @classmethod: Defines a method that operates on the class itself, not instances.
* @property: Allows you to define methods that can be accessed like attributes.

**Conclusion**

Decorators are a versatile tool in Python, making it easier to add functionality to existing code in a clean and readable way. They can greatly enhance your code's modularity and reusability!

4o mini

Top of Form

Django is a free, open-source, high-level Python web framework that enables rapid development of secure, maintainable, and scalable websites. It provides an architecture, templates, and APIs to build web applications quickly and efficiently.

Key Features:

1. Modular design

2. ORM (Object-Relational Mapping) for database interactions

3. Templates for HTML rendering

4. URL routing and dispatching

5. Authentication and authorization

6. Admin interface

7. Extensive libraries and tools

Benefits:

1. Rapid development

2. Secure (built-in protection against common web attacks)

3. Scalable

4. Maintainable

5. Flexible

6. Large community and ecosystem

Use cases:

1. Web applications (e.g., social media, e-commerce, blogs)

2. APIs (RESTful and GraphQL)

3. Microservices architecture

4. Enterprise software

5. Scientific computing and data analysis

Django's core principles:

1. Don't Repeat Yourself (DRY)

2. Convention over Configuration

3. Loose Coupling

Main components:

1. Django Project

2. Apps

3. Models

4. Views

5. Templates

6. URLs

7. Forms

8. Admin

What Is Django???

Popular Django tools and libraries:

1. Django REST framework (API development)

2. Django Channels (real-time applications)

3. Django CMS (content management)

4. Wagtail (content management)

5. Django Debug Toolbar (debugging)

Django's advantages:

1. Fast development

2. Secure by design

3. Scalable

4. Large community

5. Extensive documentation

Django's disadvantages:

1. Steep learning curve

2. Monolithic architecture

3. Not ideal for small projects

Overall, Django is a powerful and versatile framework for building complex web applications quickly and efficiently.Bottom of Form

Here's a more detailed explanation of each Django component:

1. Project

- Top-level structure of a Django application

- Created using django-admin startproject

- Contains:

- (link unavailable) (command-line utility)

- (link unavailable) (project-wide settings)

- (link unavailable) (URL configuration)

- (link unavailable) (WSGI configuration)

2. Apps

- Self-contained modules providing specific functionality

- Created using python (link unavailable) startapp

- Contain:

- (link unavailable) (database models)

- (link unavailable) (views)

- templates/ (HTML templates)

- static/ (static files)

- (link unavailable) (admin interface)

- (link unavailable) (app configuration)

3. Models

- Represent database tables

- Define data structures and relationships

- Use ORM (Object-Relational Mapping) for database interactions

- Types:

- Model (basic model)

- AbstractModel (abstract base model)

- ProxyModel (proxy model)

4. Views

- Handle HTTP requests and return responses

- Types:

- Function-based views (simple views)

- Class-based views (more complex views)

- Generic views (pre-built views)

- Use templates to render HTML

5. Templates

- Define HTML structure

- Render dynamic data from views

- Use template tags (e.g., {{ }}, {% %})

- Template inheritance and blocks

6. URLs

- Map URLs to views using URL patterns

- Use path() and re\_path() to define URL patterns

- Support URL parameters and namespaces

7. Forms

- Handle user input and validation

- Use form classes to define form structure

- Support file uploads and validation

8. Admin

- Provides a UI for model management

- Customizable using (link unavailable)

- Supports model filtering, sorting, and editing

Middleware

- Processes requests and responses

- Types:

- Request middleware (processes requests)

- Response middleware (processes responses)

- Exception middleware (handles exceptions)

Signals

- Allow apps to react to events (e.g., model saves)

- Use signal and receiver decorators

Serialization

- Converts data to/from formats (e.g., JSON)

- Use serializers module

Authentication

- Manages user authentication and permissions

- Use auth module

- Supports multiple authentication backends

Cache

- Improves performance by storing frequently accessed data

- Use cache module

- Supports multiple cache backends

These components work together to build robust, scalable web applications.

Here's a more detailed explanation of each Django component:

1. Project

- Top-level structure of a Django application

- Created using django-admin startproject

- Contains:

- (link unavailable) (command-line utility)

- (link unavailable) (project-wide settings)

- (link unavailable) (URL configuration)

- (link unavailable) (WSGI configuration)

2. Apps

- Self-contained modules providing specific functionality

- Created using python (link unavailable) startapp

- Contain:

- (link unavailable) (database models)

- (link unavailable) (views)

- templates/ (HTML templates)

- static/ (static files)

- (link unavailable) (admin interface)

- (link unavailable) (app configuration)

3. Models

- Represent database tables

- Define data structures and relationships

- Use ORM (Object-Relational Mapping) for database interactions

- Types:

- Model (basic model)

- AbstractModel (abstract base model)

- ProxyModel (proxy model)

4. Views

- Handle HTTP requests and return responses

- Types:

- Function-based views (simple views)

- Class-based views (more complex views)

- Generic views (pre-built views)

- Use templates to render HTML

5. Templates

- Define HTML structure

- Render dynamic data from views

- Use template tags (e.g., {{ }}, {% %})

- Template inheritance and blocks

6. URLs

- Map URLs to views using URL patterns

- Use path() and re\_path() to define URL patterns

- Support URL parameters and namespaces

7. Forms

- Handle user input and validation

- Use form classes to define form structure

- Support file uploads and validation

8. Admin

- Provides a UI for model management

- Customizable using (link unavailable)

- Supports model filtering, sorting, and editing

Middleware

- Processes requests and responses

- Types:

- Request middleware (processes requests)

- Response middleware (processes responses)

- Exception middleware (handles exceptions)

Signals

- Allow apps to react to events (e.g., model saves)

- Use signal and receiver decorators

Serialization

- Converts data to/from formats (e.g., JSON)

- Use serializers module

Authentication

- Manages user authentication and permissions

- Use auth module

- Supports multiple authentication backends

Cache

- Improves performance by storing frequently accessed data

- Use cache module

- Supports multiple cache backends

These components work together to build robust, scalable web applications.

Here are the main components of Django:

1. Project: The top-level structure of a Django application.

2. Apps: Self-contained modules that provide specific functionality.

3. Models: Represent database tables, defining data structures and relationships.

4. Views: Handle HTTP requests, interact with models, and return responses.

5. Templates: Define HTML structure, rendering dynamic data from views.

6. URLs: Map URLs to views using URL patterns.

7. Forms: Handle user input, validation, and processing.

8. Admin: Provides a UI for model management.

Additional Components:

1. Middleware: Processes requests and responses.

2. Signals: Allow apps to react to events.

3. Serialization: Converts data to/from formats like JSON.

4. Authentication: Manages user authentication and permissions.

5. Cache: Improves performance by storing frequently accessed data.

These components work together to build robust, scalable web applications.

    $background-color: #fff;

    $text-color: #060606;

    $hover-color: rgb(215, 35, 11);

    $main-font: "Open Sans", sans-serif;

    $header-font: "Oswald", sans-serif;

    $nav-color: #36454F;

body {

        background-color: $background-color;

        color: $text-color;

        line-height: 3em;

}

body.dark-theme{

    $background-color: #000000;

    $text-color: #60601c;

    $hover-color: aqua;

    $main-font: "Open Sans", sans-serif;

    $header-font: "Oswald", sans-serif;

    $nav-color: #d5dee4;

}

@mixin txtAlign{

    text-align: center;

}

.container1 {

    display: grid;

    grid-template-columns: 1fr 1fr 1fr .6fr;

    grid-template-rows: .5fr 1fr 2.5fr .8fr .5fr 1fr 2.5fr 2.5fr .8fr .8fr .5fr repeat(5, 1fr) .8fr .5fr 1fr repeat(6, 1.5fr) .8fr .5fr repeat(3, 1fr) 2fr;

    height: 600vh;

    grid-gap: 3px;

}

.nav1 {

    grid-area: 1/1/1/5;

    display: flex;

    flex-direction: row;

    justify-content: center;

    i {

        padding: 5px;

    }

    ul {

        list-style: none;

        justify-content: center;

        margin-top: 60px;

        li {

            display: inline;

            padding: 30px;

            color: $nav-color;

            a {

                text-decoration: none;

                font-size: 16px;

                transition: color 0.3s, font-size 0.3s;

                color: inherit;

                &:hover {

                    color: $hover-color;

                    font-size: 18px;

                    cursor: pointer;

                }

                &:focus {

                    outline: 2px dashed $hover-color; // Highlight focused elements

                }

            }

        }

    }

}

#header {

    grid-area: 2/1/3/5;

}

#intro {

    grid-area: 3/2/4/5;

    color: $text-color;

    padding: 10px;

    h1,

    .quotes {

        font-family: $header-font;

        font-weight: 400;

        margin-bottom: 0;

    }

    h3 {

        font-family: $header-font;

        font-weight: 200;

        font-variation-settings: "wdth" 100;

    }

    p {

        @extend h3;

        margin-top: 20px;

    }

}

#img1 {

    grid-area: 3/1/4/2;

    &::before {

        content: "";

        position: absolute;

        width: 50px;

        height: 150px;

        border-top: 5px solid red;

        border-left: 5px solid red;

        left: 10px;

    }

    &::after {

        content: "";

        position: absolute;

        width: 50px;

        height: 150px;

        border-bottom: 5px solid red;

        border-right: 5px solid red;

        left: 250px;

        top: 500px;

    }

}

.quotes {

    h2 {

        font-family: $main-font;

        font-weight: 400;

        font-variation-settings: "wdth" 100;

        color: rgb(243, 134, 134);

        padding: 20px;

        font-size: 35px;

    }

}

.author {

    color: $nav-color;

    text-decoration: solid underline;

}

.name{

    color: #32CD32;

    text-decoration: underline;

}

.intro {

    p {

        font-size: 25px;

        font-style: normal;

        font-size: 30px;

    }

}

#theme-toggle {

    width: 30px;

    height: 30px;

    cursor: pointer;

    position: relative;

    top: 60px;

    left: 250px;

}

#hirebtn {

    margin-left: 40px;

    height: 40px;

    width: 150px;

    border-radius: 50%;

    background-color: rgb(243, 134, 134);

    color: rgb(12, 11, 12);

    border: none;

    transition: 0.3s;

    font-size: 20px;

    &:hover {

        background-color: $hover-color;

    }

}

.footer {

    grid-area: 5/1/6/5;

}

.title1 {

    grid-area: 6/1/7/5;

    padding-left: 30px;

    h3 {

        font-size: 40px;

        font-family: $header-font;

        font-weight: 400;

        @include txtAlign();

    }

}

.text1 {

    grid-area: 7/1/8/5;

    padding-left: 20px;

    text-wrap: wrap;

    font-size: 30px;

}

.info1 {

    grid-area: 8/1/9/3;

    padding-left: 30px;

    padding-top: 0px;

    font-size: 30px;

}

.info11 {

    grid-area: 8/3/9/5;

    padding-left: 30px;

    padding-top: 80px;

    font-size: 30px;

}

.cert {

    grid-area: 9/1/10/3;

    padding-left: 30px;

    font-size: 30px;

}

.skills {

    grid-area: 9/3/10/5;

    padding-left: 30px;

    font-size: 30px;

}

.buttons {

    grid-area: 11/1/12/5;

    button {

        margin: 10px 20px;

        padding: 5px;

        height: 30px;

        width: 150px;

        background-color: rgb(52, 83, 3);

        color: white;

        transition: color 0.3s, font-size 0.3s;

        &:hover {

            background-color: $hover-color;

            color: black;

        }

    }

}

.eduexp {

    grid-area: 10/1/11/5;

    display: grid;

    grid-template-columns: 1fr 1fr;

    grid-template-rows: 2fr;

    background-color: coral;

    font-size: 30px;

}

.footer1 {

    grid-area: 11/1/12/5;

}

.title3 {

    grid-area: 12/1/13/5;

    margin-top: 20px;

    h2 {

        font-size: 40px;

        font-family: $header-font;

        padding-left: 30px;

        @include txtAlign();

    }

}

h3 {

    font-size: 25px;

    font-family: $header-font;

    padding-left: 30px;

    margin-top: 10px;

    margin-bottom: 0px;

}

p {

    font-family: $main-font;

    font-weight: 100;

    font-variation-settings: "wdth" 750;

    font-size: 30px;

    padding: 0px 20px;

    margin: 0px;

}

.info {

    grid-area: 13/1/14/5;

    font-size: 30px;

}

.descrp1, .descrp2, .descrp3, .descrp4 {

    border: 3px solid wheat;

    border-radius: 30px;

}

.descrp1 {

    grid-area: 14/1/15/3;

    font-size: 30px;

}

.descrp2 {

    grid-area: 15/3/16/5;

    font-size: 30px;

}

.descrp3 {

    grid-area: 16/1/17/3;

    font-size: 30px;

}

.descrp4 {

    grid-area: 17/3/18/5;

    font-size: 30px;

}

.footer2 {

    grid-area: 18/1/19/5;

}

.title4 {

    grid-area: 19/1/20/5;

    background-color: rgb(187, 240, 240);

    @include txtAlign();

}

.info5 {

    grid-area: 20/1/21/5;

    background-color: rgb(4, 187, 187);

    font-size: 30px;

}

.pro1 {

    grid-area: 21/1/22/5;

    background-color: rgb(112, 125, 145);

}

.pro2 {

    grid-area: 22/1/23/5;

    background-color: rgb(49, 115, 159);

}

.pro3 {

    grid-area: 23/1/24/5;

    background-color: rgb(128, 137, 225);

}

.pro4 {

    grid-area: 24/1/25/5;

    background-color: rgb(174, 205, 243);

}

.pro5 {

    grid-area: 25/1/26/5;

    background-color: rgb(51, 58, 119);

}

.pro6 {

    grid-area: 26/1/27/5;

    background-color: rgb(72, 154, 255);

}

.footer3 {

    grid-area: 27/1/28/5;

}

.subtitle, .subtitle2 {

    text-align: center;

    margin-top: 20px;

    font-size: 35px;

}

.title5 {

    grid-area: 28/1/29/5;

    text-align: center;

    h2 {

        @include txtAlign();

    }

}

.subtitle {

    grid-area: 29/1/30/5;

}

.contact {

    grid-area: 30/1/31/5;

    display: grid;

    grid-template-columns: .5fr .5fr .5fr .5fr;

    grid-template-rows: auto;

    .call, .mail, .github, .linked {

        padding: 20px;

    }

    i:hover {

        color: aqua;

        font-size: 18px;

    }

}

.subtitle2 {

    grid-area: 31/1/32/5;

    font-size: 35px;

}

.form {

    grid-area: 32/1/33/5;

}

input {

    margin: 20px;

    height: 40px;

    width: 400px;

}

textarea {

    margin: 20px;

    width: 400px;

    height: 50px;

}

button {

    margin: 20px;

    width: 200px;

    height: 30px;

}

@media only screen and (max-width: 550px) {

    .container1 {

        grid-template-columns: 1fr; // Stack all columns

        grid-template-rows: repeat(45, 1fr); // Adjust rows to fit single column

        height: auto; // Adjust height to content

    }

    @media only screen and (max-width: 550px) {

        .nav1 {

            flex-direction: column; // Stack navigation items vertically

            ul {

                list-style-type: disc; // Use bullet points for list items

                padding-left: 20px; // Add some left padding for bullet alignment

                margin-top: 20px; // Reduce margin

                li {

                    display: list-item; // Ensure items are displayed as list items

                    padding: 10px; // Adjust padding for smaller screens

                    text-align: left; // Align text to the left

                    a {

                        text-decoration: none; // Keep links unstyled

                        color: aliceblue; // Text color

                        transition: color 0.3s, font-size 0.3s;

                        &:hover {

                            color: $hover-color;

                            font-size: 18px;

                            cursor: pointer;

                        }

                        &:focus {

                            outline: 2px dashed $hover-color; // Highlight focused elements

                        }

                    }

                }

            }

        }

    }

    #icon {

        top: 20px; // Adjust position for smaller screens

        left: 50%; // Center icon

        transform: translateX(-50%); // Centering adjustment

    }

    #hirebtn {

        margin: 20px auto; // Center button

        width: 80%; // Full width on small screens

    }

    h3 {

        font-size: 20px; // Reduce font size

        padding-left: 0; // Remove padding for smaller screens

        text-align: center; // Center align text

    }

    .quotes h2 {

        font-size: 24px; // Adjust font size for smaller screens

        padding: 10px; // Reduce padding

    }

    p {

        font-size: 16px; // Adjust paragraph font size

        padding: 0 10px; // Reduce padding

    }

    input, textarea, button {

        width: 90%; // Make input elements responsive

        margin: 10px auto; // Center and adjust margin

        height: 40px; // Consistent height

    }

    .contact {

        grid-template-columns: 1fr; // Stack contact items

        .call, .mail, .github, .linked {

            padding: 10px; // Reduce padding

        }

    }

    .buttons {

        button {

            width: 100%; // Full width buttons

            margin: 10px 0; // Adjust margin

        }

    }

    .title1, .title3, .title5 {

        h2, h3 {

            font-size: 30px; // Adjust title font sizes

            padding-left: 0; // Remove padding for smaller screens

        }

    }

    .descrp1, .descrp2, .descrp3, .descrp4 {

        margin: 10px; // Adjust margins

    }

}