1. Variables and Constants

-Variables:

- Definition: Containers for values that can change during the program’s execution.
- Syntax: `var name = "John"`
- Example:

```
var name = "Alice"
name = "Bob"  // name now holds "Bob"
```

- Constants:

- Definition: Containers for values that cannot be changed once set.
- Syntax: `let age = 30`
- Example:

```
let age = 30
// age cannot be changed
```

2. Data Types

- Integer (`Int`):

- Definition: Represents whole numbers.
- Syntax: `let count: Int = 42`
- Example:

```
let score: Int = 100
```

- Floating-Point Numbers (`Double`, `Float`):

  - Definition: Represents numbers with fractional parts.

  - Syntax: `let pi: Double = 3.14159`

  - Example:

    let pi: Double = 3.14

- String (`String`):

  -Definition: Represents a sequence of characters.

  - Syntax:`let greeting: String = "Hello, Swift!"`

  - Example:

    let message = "Hello, World!"

- Boolean (`Bool`):

  - Definition: Represents true or false values.

  -Syntax: `let isActive: Bool = true`

  - Example:

    let isFinished: Bool = false

3. Control Flow

- Conditional Statements (`if`, `else if`, `else`):
  - **Definition:** Executes code based on conditions.
  - **Syntax:**

```
if condition {
    // Code to execute if condition is true
} else {
    // Code to execute if condition is false
}
```

  - **Example:**

```
let temperature = 20
if temperature > 25 {
    print("It's warm")
} else {
    print("It's cool")
}
```

- **Switch Statement:**
  - **Definition:** Provides a way to execute different code based on the value of a variable.
  - **Syntax:**

```
switch value {
```

```
  case pattern1:
     // Code
  case pattern2:
     // Code
  default:
     // Code
  }
```

- **Example:**

```
  let dayOfWeek = "Monday"
  switch dayOfWeek {
  case "Monday":
     print("Start of the week")
  case "Friday":
     print("End of the week")
  default:
     print("Midweek")
  }
```

- **Loops (`for`, `while`):**
  - **Definition:** Repeats code multiple times.
  - **For Loop Syntax:**

```
for item in collection {
    // Code
}
```

- **While Loop Syntax:**

```
while condition {
    // Code
}
```

- **Example:**

```
for i in 1...5 {
    print(i)
}

var count = 1
while count <= 5 {
    print(count)
    count += 1
}
```

4. Functions

- **Definition:** Blocks of code that perform a task and can be called with arguments.
- **Syntax:**

```
func functionName(parameters) -> ReturnType {
    // Code
    return value
}
```

- **Example:**

```
func greet(name: String) -> String {
    return "Hello, \(name)!"
}

let greeting = greet(name: "Alice")
```

5. Classes and Structures

- **Classes:**
  - **Definition:** Reference types that can have properties, methods, and initializers.
  - **Syntax:**

```
class ClassName {
    var property: Type
```

```
    init(property: Type) {
        self.property = property
    }
    func method() {
        // Code
    }
}
```

- **Example:**

```
class Person {
    var name: String
    var age: Int

    init(name: String, age: Int) {
        self.name = name
        self.age = age
    }

    func describe() -> String {
        return "\(name) is \(age) years old."
    }
}

let person = Person(name: "John", age: 30)
```

- **Structures:**
  - **Definition:** Value types similar to classes but with no inheritance.
  - **Syntax:**

```
struct StructName {
    var property: Type
}
```

  - **Example:**

```
struct Point {
    var x: Int
    var y: Int
}

let point = Point(x: 10, y: 20)
```

6. Optionals

- **Definition:** A type that can hold either a value or `nil` to indicate the absence of a value.
- **Syntax:**

```
var optionalValue: Type? = nil
```

- **Example:**

```
var name: String? = "John"
if let unwrappedName = name {
    print("Name is \(unwrappedName)")
} else {
    print("Name is nil")
}
```

7. Arrays and Dictionaries

- **Arrays:**
  - **Definition:** Ordered collections of values.
  - **Syntax:**

```
var array: [Type] = [value1, value2]
```

  - **Example:**

```
var numbers = [1, 2, 3, 4, 5]
numbers.append(6)
```

- **Dictionaries:**
  - **Definition:** Unordered collections of key-value pairs.

- **Syntax:**

```
var dictionary: [KeyType: ValueType] = [key1: value1, key2: value2]
```

- **Example:**

```
var person = ["name": "John", "age": "30"]
let name = person["name"]
```

8. Error Handling

- **Definition:** Mechanism to handle unexpected conditions or errors in code.
- **Syntax:**

```
enum CustomError: Error {
    case somethingWentWrong
}

func doSomething() throws {
    throw CustomError.somethingWentWrong
}

do {
    try doSomething()
```

```
} catch CustomError.somethingWentWrong {
    print("Handled error")
}
```

- **Example:**

```
enum FileError: Error {
    case fileNotFound
}

func readFile(filename: String) throws -> String {
    if filename != "existingFile.txt" {
        throw FileError.fileNotFound
    }
    return "File content"
}

do {
    let content = try readFile(filename: "missingFile.txt")
    print(content)
} catch FileError.fileNotFound {
    print("File not found")
}
```

1. Range Types

Closed Range (`a...b`)

- **Definition:** A closed range includes both the lower and upper bounds. It's used when you want to include both endpoints in your range.

- **Syntax:** `a...b`

- **Example:**

```
let closedRange = 1...5
// Represents the values: 1, 2, 3, 4, 5

for i in closedRange {
    print(i)
}
// Output: 1, 2, 3, 4, 5
```

Half-Open Range (`a..<b`)

- **Definition:** A half-open range includes the lower bound but excludes the upper bound. It's often used in loops and array slicing where the end value is not included.

- **Syntax:** `a..<b`

- **Example:**

```
let halfOpenRange = 1..<5
```

```
// Represents the values: 1, 2, 3, 4

for i in halfOpenRange {
    print(i)
}
// Output: 1, 2, 3, 4
```

2. Using Ranges

Iteration

- **Definition:** You can use ranges to iterate over a sequence of values in loops.
- **Example:**

```
for number in 1...3 {
    print(number)
}
// Output: 1, 2, 3
```

Membership Testing

- **Definition:** You can check if a value falls within a range using the `contains(_:)` method.
- **Example:**

```
let number = 4
if (1...5).contains(number) {
    print("\(number) is within the range")
} else {
    print("\(number) is outside the range")
}
// Output: 4 is within the range
```

Array Slicing

- **Definition:** Ranges can be used to slice arrays and access subsets of elements.
- **Example:**

```
let array = [10, 20, 30, 40, 50]
let slice = array[1...3]
print(slice)
// Output: [20, 30, 40]
```

Character Ranges

- **Definition:** Ranges can be used with characters to handle sequences of characters.
- **Example:**

```
let vowelRange: ClosedRange<Character> = "a"..."u"
```

```
 for char in "a"..."z" {
    if vowelRange.contains(char) {
       print(char)
    }
 }
 // Output: a, e, i, o, u
```

Variadic Parameter:

Definition: A parameter that can accept zero or more values of a specific type. It is indicated by appending ... after the parameter's type.

Usage: It allows you to pass a list of values to a function without needing to create an array explicitly.

```
func printNumbers(_ numbers: Int...) {
   for number in numbers {
      print(number)
   }
}
```