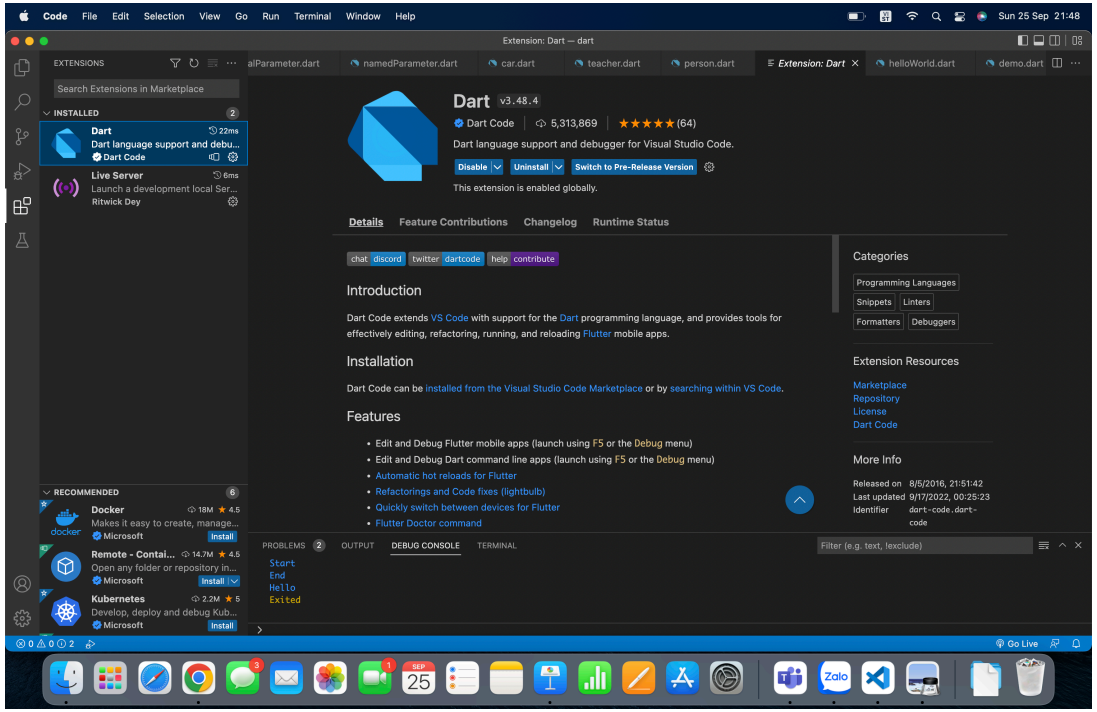
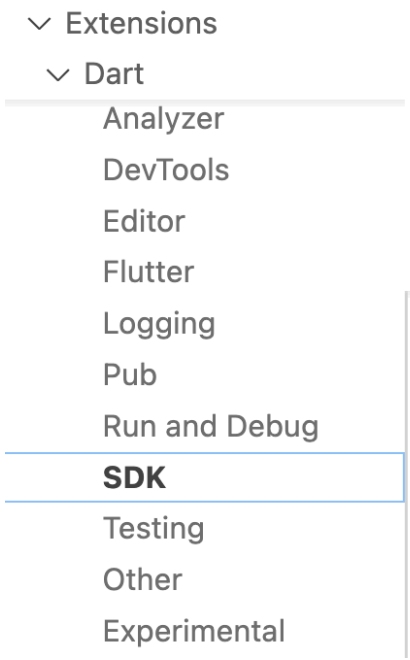
Resource Software Solution

Dart Language

Contents

Visual Studio Code IDE





Extensions

Dart

SDK

Dart: Sdk Path *(Also modified in Workspace)*

The location of the Dart SDK to use for analyzi
attempt to find it from the `PATH` environment v
Flutter SDK is used in preference.

Edit in settings.json

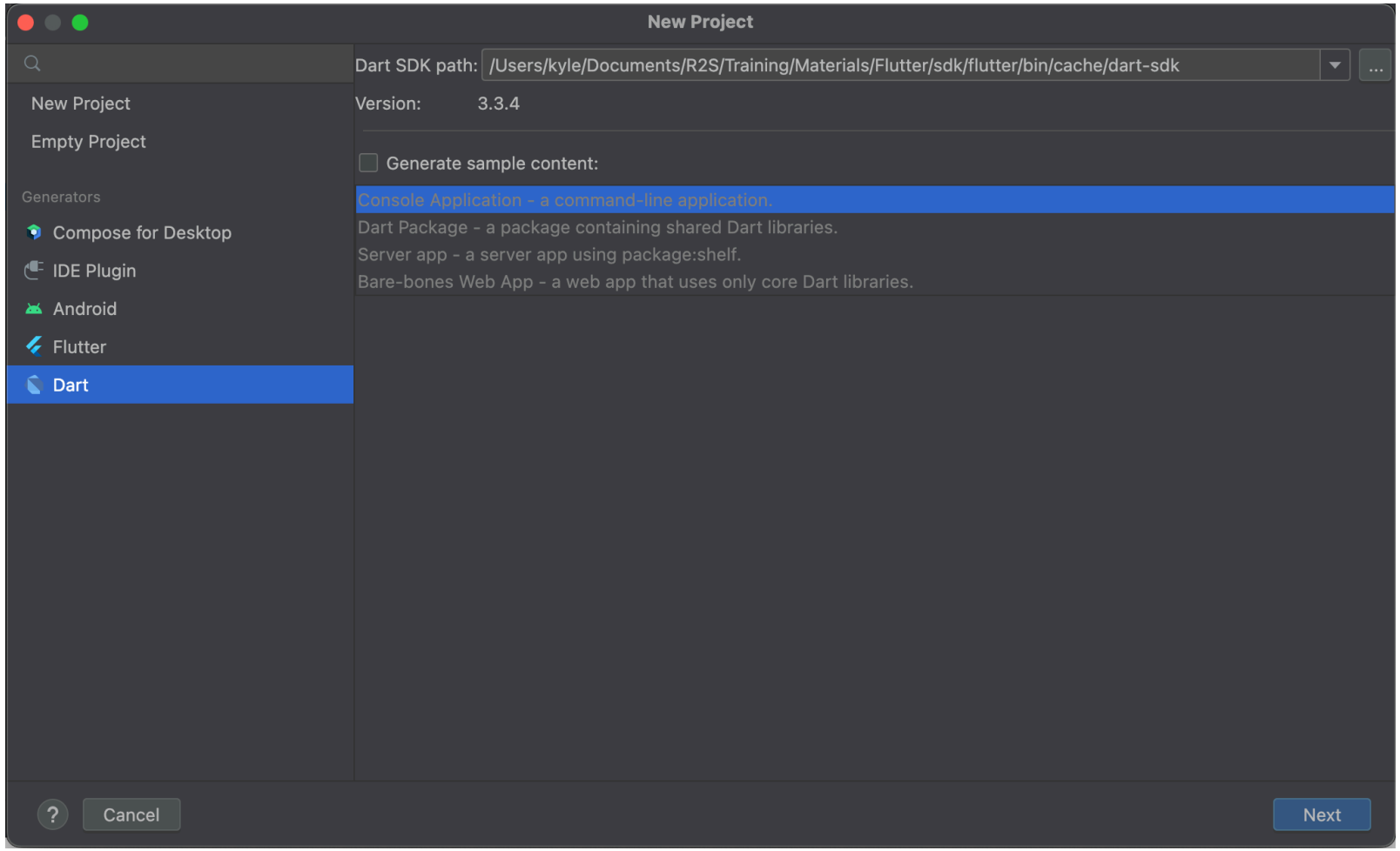
settings.json

Users > kyle > Library > Application Support > Code > User > {} settings.json > ...

```
1  {
2      "editor.fontSize": 16,
3      "security.workspace.trust.untrustedFiles": "open",
4      "editor.minimap.enabled": false,
5      "dart.sdkPath": "/Users/kyle/Documents/R2S/Training/Materials/Flutter/Dart language/dart-sdk",
6      "dart.cliConsole": "terminal",
7      "workbench.colorTheme": "Default Light+",
8      "editor.accessibilitySupport": "on",
9      "code-runner.runInTerminal": true,
10     "window.zoomLevel": 1,
11     "dart.flutterSdkPath": null,
12     "explorer.confirmDragAndDrop": false
13 }
```

Android Studio IDE

- File -> New -> New Flutter Project



/Users/Kyle/Documents/R2S/Training/Materials/Flutter/sdk/flutter/bin/cache/dart-sdk

General

- Naming convention: **lowerCamelCase**, the first word is always all lowercase letters, including the first letter. The proceeding words are all capitalized -- only the first letter of each word is an uppercase letter.
- Comment code: like C
- main() entry point

```
1  void main() {
2    // Incorrect Way
3    var fullname = "John Doe";
4    // Correct Way
5    var fullName = "John Doe";
6    const pi = 3.14;
7
8    print(fullName);
9  }
```

Variables (1)

- Assume you want to store two values 10 and 20 in your program and at a later stage, you want to use these two values.
- In programming, you need to **manage values like numbers, strings, and booleans**. **To store these values in programs, you use variables.**
- A variable is an **identifier** that stores a value of a specific type.
- By definition, **a variable** is associated with a **type** and has a **name**.

Variables (2)

- Declaring variables **type variableName [= initializing];**
- Example **String name = "John";**
- Types

1. **String**: For storing text value. E.g. “John” [Must be in quotes]
2. **int**: For storing integer value. E.g. 10, -10, 8555 [Decimal is not included]
3. **double**: For storing floating point value. E.g. 10.0, -10.2, 85.698 [Decimal is included]
4. **num**: For storing any types of number. E.g. 10, 20.2, -20 [both int and double]
5. **bool**: For storing true or false value. E.g. true, false [Only stores true or false values]
6. **var**: For storing any value. E.g. ‘Bimal’, 12, ‘z’, true
7. **dynamic**

Variables (3)

- Example

```
void main() {
    // Declaring Variables
    String name = "John";
    num age = 20; // used to store any types of numbers
    num height = 5.9;
    bool isMarried = false;

    // printing variables value
    print("Name is $name");
    print("Age is $age");
    print("Height is $height");
    print("Married Status is $isMarried");
}
```

Syntax: type variableName [= value];

Data Types (1)

- Numbers

```
int counter = 0;
double price = 0.0;
price = 125.00;
```

- Strings

```
// Strings
String defaultMenu = 'main';

// String concatenation
String combinedName = 'main' + ' ' + 'function';
String combinedNameNoPlusSign = 'main' ' ' 'function';

// String multi-line
String multilineAddress = '''
  123 Any Street
  City, State, Zip
''';
```

Data Types (2)

- Booleans

```
// Booleans
bool isDone = false;
isDone = true;
```

- Lists ~ Arrays

```
// Lists - In Dart List is an array
List listOfFilters = ['company', 'city', 'state'];
listOfFilters.forEach((filter) {
   print('filter: $filter');
});
// Result from print statement
// filter: company
// filter: city
// filter: state
```

Data Types (3)

- Maps

```
// Maps - An object that associates keys and values.
// Key: Value - 'KeyValue': 'Value'
Map mapOfFilters = {'id1': 'company', 'id2': 'city', 'id3': 'state'};

// Change the value of third item with Key of id3
mapOfFilters['id3'] = 'my filter';

print('Get filter with id3: ${mapOfFilters['id3']}');
// Result from print statement
// Get filter with id3: my filter
```

Data Types (4)

- var keyword: **var** automatically finds a data type.
- In simple terms, **var says if you don't want to specify a data type, i will find a data type for you**.

```
void main() {
    var name = "John Doe";  // String
    var age = 20;           // int

    print(name);
    print(age);
}
```

Data Types (5)

- Dynamically typed

```
void main() {
  dynamic myVariable = 50;
  myVariable = "Hello";
  print(myVariable);
}
```

dynamic
vs var

- var typed

```
void main() {
  var myVariable = 50; // You can also use int instead of var
  myVariable = "Hello"; // this will give error
  print(myVariable);
}
```

Constant (1)

- Constants are variable types that **cannot be changed value after initialization**.
- Defining/Initializing Constant in Dart
  - Using the **final** keyword
  - Using the **const** keyword
- Syntax

```
final const_name;
Or
final data_type const_name;
```

```
const const_name;
Or
const data_type const_name;
```

Constant (2)

- Let's understand the following example

```
void main () {
  final age = 18;
  const name = "Peter";

  print(age);
  print(name);
}
```

difference between the "const" and "final"

Constant (3)

- The "const" vs "final" keywords in Dart
- const keyword:
  - **Value must be known at compile-time**
  - **const birthday = "2008/12/25"**
  - Can't be changed after initialized.
- final keyword:
  - **Value must be known at run-time**
  - **final birthday = getBirthDateFromDB();**
  - Can't be changed after initialized.

```
void main () {
  final age;      // OK
  const name; // Error

  print(age);
  print(name);
}
```

Operators

1. Arithmetic operators
2. Increment and Decrement operators
3. Assignment operators
4. Relational operators
5. Logical operators
6. Type Test operators

Arithmetic Operators (1)

Operator Symbol	Operator Name	Description
+	Addition	For adding two operands
-	Subtraction	For subtracting two operands
-expr	Unary Minus	For reversing the sign of the expression
*	Multiplication	For multiplying two operands
/	**Division**	**For dividing two operands and give output in double**
~/	**Division**	**For dividing two operands and give output in integer**
%	Modulus	Remainder After Integer Division
++	Increment	Increase Value By 1. For E.g a++;
--	Decrement	Decrease Value By 1. For E.g a–;

Arithmetic Operators (2)

- Example

```
void main() {
  // declaring two numbers
  int num1=10;
  int num2=3;

  // performing arithmatic calulation
  int sum=num1+num2;        // addition
  int diff=num1-num2;       // subtraction
  int unaryMinus = -num1;     // unary minus
  int mul=num1*num2;        // multiplication
  double div=num1/num2;     // division
  int div2 =num1~/num2;      // integer division
  int mod=num1%num2;        // show remainder

  //Printing info
  print("The addition is $sum.");
  print("The subtraction is $diff.");
  print("The unary minus is $unaryMinus.");
  print("The multiplication is $mul.");
  print("The division is $div.");
  print("The integer division is $div2.");
  print("The modulus is $mod.");
}
```

Increment/Decrement Operators (1)

Operator Symbol	Operator Name	Description
++var	Pre Increment	Increase Value By 1. var = var + 1 **Expression value is var+1**
--var	Pre Decrement	Decrease Value By 1. var = var - 1 **Expression value is var-1**
var++	Post Increment	Increase Value By 1. var = var + 1 **Expression value is var**
var--	Post Decrement	Decrease Value By 1. var = var - 1 **Expression value is var**

Increment/Decrement Operators (2)

- Example

```
1   void main() {
2     // declaring two numbers
3     int num1 = 0;
4     int num2 = 0;
5
6     // performing increment / decrement operator
7
8     // pre increment
9     num2 = ++num1;
10    print("The value of num2 is $num2");
11
12    // reset value to 0
13    num1 = 0;
14    num2 = 0;
15
16    // post increment
17    num2 =  num1++;
18    print("The value of num2 is $num2");
19
20  }
```

Assignment Operators

Operator Type	Description
=	Assign a value to a variable
+=	Adds a value to a variable
-=	Reduces a value to a variable
*=	Multiply value to a variable
/=	Divided value by a variable

```
void main() {
  double age = 24;
  age+= 1;  // Here age+=1 means age = age + 1.
  print("After Addition Age is $age");
  age-= 1;  //Here age-=1 means age = age - 1.
  print("After Aubtraction Age is $age");
  age*= 2;  //Here age*=2 means age = age * 2.
  print("After Multiplication Age is $age");
  age/= 2;  //Here age/=2 means age = age / 2.
  print("After Division Age is $age");
}
```

Relational Operators

Operator Symbol	Operator Name	Description
>	Greater than	Used to check which operand is bigger and gives result as boolean
<	Less than	Used to check which operand is smaller and gives result as boolean
>=	Greater than or equal to	Used to check which operand is bigger or equal and gives result as boolean
<=	Less than or equal to	Used to check which operand is smaller or equal and gives result as boolean
==	Equal to	Used to check operands are equal to each other and gives result as boolean
!=	Not equal to	Used to check operand are not equal to each other and gives result as boolean

Logical Operators

Operator Type	Description
&&	This is ‘and’, return true if all conditions are true
||	This is ‘or’. Return true if one of the conditions is true
!	This is ’not’. return false if the result is true and vice versa

```
void main() {
  int userid = 123;
  int userpin = 456;

   // Printing Info
   print((userid == 123) && (userpin== 456));    // print true
   print((userid == 1213) && (userpin== 456)); // print false.
   print((userid == 123) || (userpin== 456));      // print true.
   print((userid == 1213) || (userpin== 456));    // print true
   print((userid == 123) != (userpin== 456));     //print false
}
```

Type Test Operators

- Type test operators are useful for checking types at runtime.

Operator Symbol	Operator Name	Description
is	is	Gives boolean value true if the object has a specific type
is!	is not	Gives boolean value false if the object has a specific type

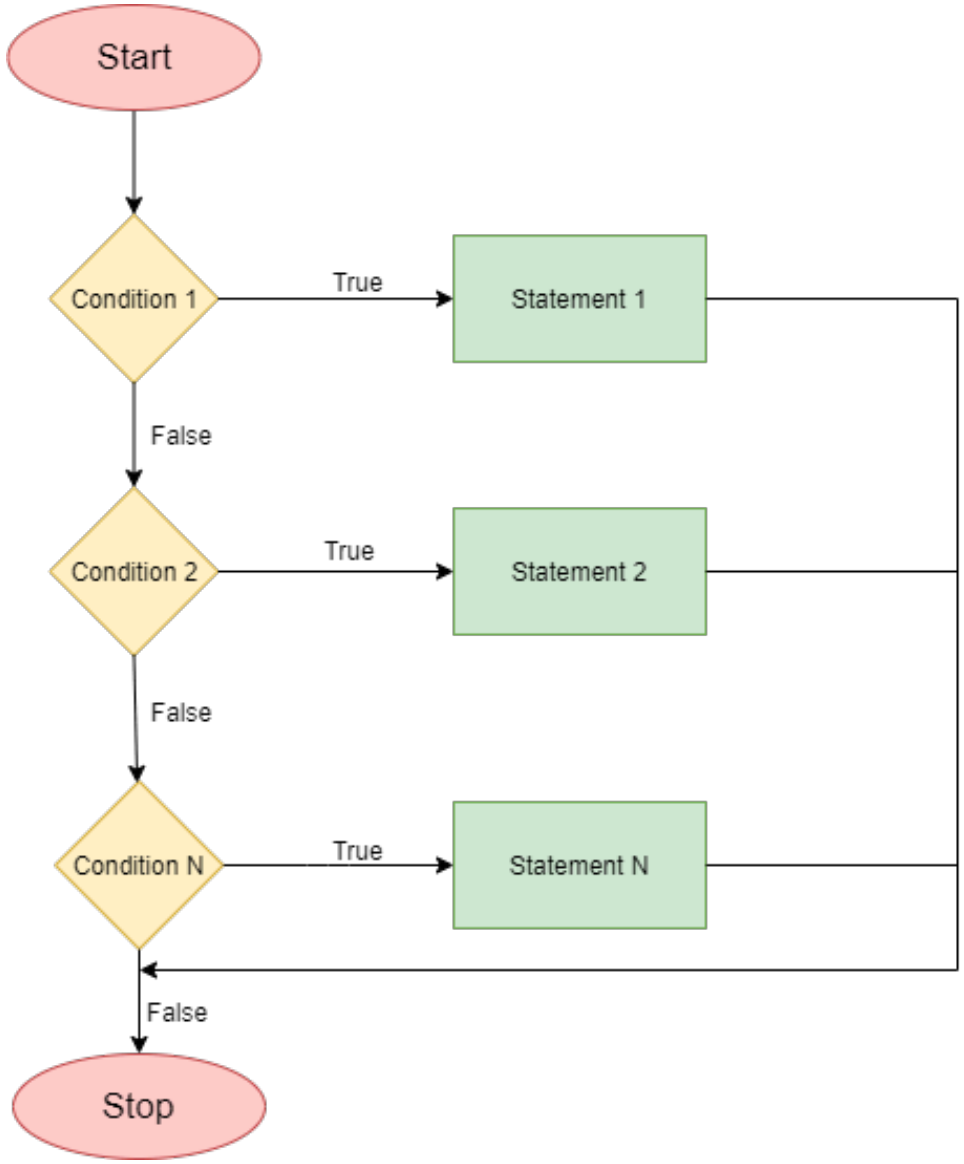
```
void main() {
  String value = "Dart Tutorial";
  double gpa = 10;

  print(value is String); //true
  print(gpa is !double); //false
}
```

Condition in Dart (1)

- if else

```
if (condition1) {
   statements1;
} else if (condition2) {
   statements2;
} else if (condition3) {
  statements3;
}
.
.
else {
   statementsN;
}
```



Condition in Dart (2)

- if else

```
if (condition1) {
  statements1;
} else if (condition2) {
  statements2;
} else if (condition3) {
 statements3;
}
.
.
else {
  statementsN;
}
```

```
1   void main() {
2     var marks = 74;
3     if (marks > 85) {
4       print("Excellent");
5     } else if (marks > 75) {
6       print("Very Good");
7     } else if (marks > 65) {
8       print("Good");
9     } else {
10      print("Average");
11    }
12  }
```

Condition in Dart (3)

- Ternary operator **condition ? exprIfTrue : exprIfFalse**
- If Else vs Ternary Operator

```
void main() {
  int num1 = 10;
  int num2 = 15;
  int max = 0;
  if (num1> num2) {
    max = num1;
  } else {
    max = num2;
  }
  print("The greatest number is $max");
}
```

```
void main() {
  int num1 = 10;
  int num2 = 15;
  int max = (num1 > num2) ? num1 : num2;
  print("The greatest number is $max");
}
```

Condition in Dart (4)

- switch case

```
switch (expression) {
    case value1:
        // statements
        break;
    case value2:
        // statements
        break;
    case value3:
        // statements
        break;
    default:
        // default statements
}
```

```dart
void main() {
  var dayOfWeek = 5;
  switch (dayOfWeek) {
    case 1:
      print("Day is Sunday.");
      break;
    case 2:
      print("Day is Monday.");
      break;
    case 3:
      print("Day is Tuesday.");
      break;
    case 4:
      print("Day is Wednesday.");
      break;
    case 5:
      print("Day is Thursday.");
      break;
    case 6:
      print("Day is Friday.");
      break;
    case 7:
      print("Day is Saturday.");
      break;
    default:
      print("Invalid Weekday.");
      break;
  }
}
```

Loops in Dart (1)

- while

```
while (condition) {
    // statement(s);
}
```

- do-while

```
do {
  // statement(s)
} while (condition);
```

Example

```
void main() {
  int i = 1;
  while (i <= 10) {
    print(i);
    i++;
  }
}
```

```
void main() {
  int i = 1;
  do {
    print(i);
    i++;
  } while (i <= 10);
}
```

Loops in Dart (2)

- for

```
// Standard for loop
List listOfFilters = ['company', 'city', 'state'];
for (int i = 0; i < listOfFilters.length; i++) {
   print('listOfFilters: ${listOfFilters[i]}');
}
// Result from print statement
// listOfFilters: company
// listOfFilters: city
// listOfFilters: state
```

- for-in

```
// or for-in loop
List listOfNumbers = [10, 20, 30];
for (int number in listOfNumbers) {
   print('number: $number');
}
// Result from print statement
// number: 10
// number: 20
// number: 30
```

Loops in Dart (3)

#	For loop	While loop	Do while loop
1	If the condition is not true first time than control will never enter in a loop	If the condition is not true first time than control will never enter in a loop.	Even if the condition is not true for the first time the control will enter in a loop.
2	Initialization and updating is the part of the syntax.	Initialization and updating is not the part of the syntax.	Initialization and updating is not the part of the syntax
3	For loop is use when we know the number of iterations means where the loop will terminate.	While loop is use when we don't know the number of iterations means where the loop will terminate.	Do while loop is use when we don't know the number of iterations means where the loop will terminate.

break Statement

- Sometimes you will need to break out of the loop immediately without checking the condition. You can do this using **break statement**.

```
void main() {
  for (int i = 1; i <= 10; i++) {
    if (i == 5) {
      break;
    }
    print(i);
  }
}
```

```
while (testExpression) {
    // codes
    if (condition to break) {
        break;
    }
    // codes
}
```

```
do {
    // codes
    if (condition to break) {
        break;
    }
    // codes
}
while (testExpression);
```

```
for (init; testExpression; update) {
    // codes
    if (condition to break) {
        break;
    }
    // codes
}
```

continue Statement

- Sometimes you will need to skip an iteration for a specific condition. You can do this utilizing **continue statement**.

```
void main() {
  for (int i = 1; i <= 10; i++) {
    if (i == 5) {
      continue;
    }
    print(i);
  }
}
```

```
while (testExpression) {
    // codes
    if (testExpression) {
        continue;
    }
    // codes
}
```

```
do {
    // codes
    if (testExpression) {
        continue;
    }
    // codes
}
while (testExpression);
```

```
for (init; testExpression; update) {
    // codes
    if (testExpression) {
        continue;
    }
    // codes
}
```

*Keeping up those **inspiration** and the **enthusiasm** in the **learning path**. Let confidence to bring it into **your career path** for getting gain the **success** as your expectation.*

Thank you

Contact

- Name: R2S Academy
- Email: daotao@r2s.edu.vn
- Phone/Zalo: 0919 365 363
- FB: https://www.facebook.com/r2s.tuyendung
- Website: www.r2s.edu.vn

Questions and Answers