Basic Programming Principles

While teaching you to code, we will be using a **programming language** called **Dart**. Dart is primarily used in app development, but may also be used for web development, video game creation, basic scripting, and more. Other programming languages of interest include Java, JavaScript, Python, and BASIC.

To code using dart, go to dartpad.dev, where you'll find an online code editor. There will already be sample code written—feel free to delete it. Also, try not to use the AI code assistant. The point is to learn to code, not to learn how to watch AI do it.

The Basics

One of the most important concepts in Computer Science is the **function**: a set of instructions for a computer to follow. In fact, your computer will read your **program** as a function itself.

Dart runs each program through the ***main* function**. By writing the following code, you have successfully defined a program for the computer to run.

```
1 void main() {}
```

...it doesn't do anything though.

The easiest way to visibly accomplish something through a program is by using the **print command**. Write the following code, including:

print()	Calls the print command to run.
"Hello World!"	The quotation marks specifies that the letters within serve as text. By placing it within the parentheses of the print command, it serves as an **input**.
;	The **semicolon** is essential—it tells the computer where an instruction ends.

...all together to get:

```
1 void main() {
2   print("Hello World!");
3 }
```

Output: Hello World!

However, this code is a bit clunky. Hypothetically, an entire program could be written to occupy only one line. However, it's typically easier on the eye to split it into multiple lines.

One way we can do this is by utilizing **variables**. By defining a variable, your computer will remember a value associated with it. A variable can be declared in the following manner:

```
1 void main() {
2   var text = "Hello World!";
3   print(text);
4 }
```

Output: Hello World!

As you can see, the code operates just as if the value had been directly plugged into the function.

More on the print function... it doesn't just print text! In fact, the print function takes any form of input.

Another useful thing you can display through the print function is a **number**. In most programming languages, a number is not only just a basic **digit**, but also the result of an **operation**. As a result, not only is 4 classified as a number, but so is 2+2.

```
1 void main() {
2   var text = "Hello World!";
3   print(2+2);
4 }
```

Output: 4

As you can see, the computer calculates the sum before printing.

In the image above, you can see there's a yellow underline below our text variable. This is an example of a **code warning**. A code warning could be anything from a minor syntax mistake to a code-breaking error. Because this one only has a yellow underline, we can tell

that it's only a minor warning: the variable *text* is unused. While it would be more optimal to remove this unused variable, the program still runs with the inclusion of it.

Back to the subject of variables, they can also have numbers as values. Thus, the two separate instances of 2 can be substituted for variables like so:

```
1 void main() {
2   var text = "Hello World!";
3
4   var a = 2;
5   var b = 2;
6   print(a+b);
7 }
```

Output: 4

As you can see, everything is still calculated and printed properly. Considering that 2+2 itself is an instance of a number, we could simplify the code to the following:

```
1 void main() {
2   var text = "Hello World!";
3
4   var a = 2 + 2;
5   print(a);
6 }
```

Output: 4

However, this is an eyesore. While it would be easier to simply to *a* = 4, we could also split this operation into multiple lines.

It should be noted that almost all variables can be reassigned a value after they are reassigned by using an **assignment operator** (AKA an equals sign). Try redefining *a* to 6 using the following code as an example:

```
1 void main() {
2   var text = "Hello World!";
3
4   var a = 2 + 2;
5   a = 6;
6   print(a);
7 }
```

Output: 6

Notice how, when redefining *a*, "var" was omitted. That's because "var" is used to specify the declaration of a variable—when a variable already exists, it cannot be re-defined, only re-assigned.

Since *a* was reassigned after its initial declaration, the value printed is 6. If you haven't noticed yet, code runs from top to bottom order. If we were to reassign *a* after it's printed, just like done in the following code:

```
1 void main() {
2   var text = "Hello World!";
3
4   var a = 2 + 2;
5   print(a);
6   a = 6;
7 }
```

Output: 4

Then *a* is still equal to 4 when printed and is not reassigned until after printing.

In addition to redefining a variable through an assignment operator, the value of a variable can also be altered through an **augmented assignment operator**. For example, we could split the assignment of *a* as 2+2 by initially defining *a* as 2 and adding 2 in the next line, just like in the following code:

```
1 void main() {
2   var text = "Hello World!";
3
4   var a = 2;
5   a += 2;
6   print(a);
7 }
```

Output: 4

In addition to... addition, you can also subtract a value. While you could simple add a negative value to a number, you can also substitute the "+" in the augmented assignment operator for a "-", just like below:

```
1 void main() {
2   var text = "Hello World!";
3
4   var a = 2;
5   a -= 2;
6   print(a);
7 }
```

Output: 0

In addition to using "-" as part of the augmented assignment operator, you can also use "-" in defining a number, just like any other operator:

```
1 void main() {
2   var text = "Hello World!";
3
4   var a = 2;
5   a -= 2-4;
6   print(a);
7 }
```

Output: 4

As well as addition and subtraction, you can also multiply and divide values both through definition and an augmented assignment operator.

In most programming languages, multiplication is done through a "*". Thus, if we were to substitute the "-" for a "*" in the code like so:

```
1 void main() {
2   var text = "Hello World!";
3
4   var a = 2;
5   a *= 2-4;
6   print(a);
7 }
```

Output: -4

Then *a* is multiplied by 2-4 instead of subtracted by. To divide a number, use the symbol "/":

```
1 void main() {
2   var text = "Hello World!";
3
4   var a = 2;
5   a /= 2-4;
6   print(a);
7 }
```

ERROR

...except that doesn't work.

Why? Because *a* is currently set to an **integer** and not a rational number (or in computer talk, a **double**). In most programming languages, when a variable is set as an integer, it cannot be set to anything other than an integer. Although the result, -1.0, is an integer, division does not always result in an integer, and thus the computer does not except than an integer be divided. To fix this issue, simply specify *a* as a rational number / double by adding a decimal point to its value, like so:

```
1 void main() {
2   var text = "Hello World!";
3
4   var a = 2.0;
5   a /= 2-4;
6   print(a);
7 }
```

Output: -1.0

Now, everything works fine.

However, I bet the differences between integers and doubles can be very confusing—it's even confusing to the computer!

So far, we've been defining variables using the "var" keyword. This allows us to define a variable without specifying its type, leaving the computer to figure out what kind of value it is. While this makes coding quicker, it makes it ambiguous too. It's best practice to specify the **type** of a variable of definition.

So far, we've been working with variables as text and numbers, though we've figured out that there's more than one type of number: integers and doubles. To specify the type of a

variable on declaration, substitute the “var” keyword for “int” or “double” for integers or doubles respectively.

Also, text is not referred to as “text” in programming. A variable which has text for a value is called a **String**, as it contains a string of characters to form text. Thus, we can properly define a String by substituting “var” for “String”. With all of this, we can alter the code to this:

```
void main() {
  String text = "Hello World!";

  double a = 2.0;
  a /= 2-4;
  print(a);
}
```

Output: -1.0

And it works just the same, it’s just a little clearer to both the computer and us.

By now, we’ve had a basic introduction to functions, variables, and operations. Now, it’s time to learn **conditionals**. A conditional is a set of code which runs only on a certain condition. An example of a conditional is seen below:

```
void main() {
  String text = "Hello World!";

  if(true) {
    print(text);
  }
}
```

Output: Hello World!

As you can see, a conditional is written by writing “if”, followed by a condition wrapped in parentheses, followed by the code to run on the condition wrapped in curly braces. Every conditional is basically checking if the specified condition is true. Here, I used the literal value of “true”, meaning that the code within the conditional will always run. Alternatively, if I were to use “false”:

```
void main() {
  String text = "Hello World!";

  if(false) {
    print(text);
  }
}
```

No Output

The program never prints, and it never will. Even the computer recognizes this—it gives us a code warning on the conditional, as it's what's referred to as **dead code**—code which will never run.

You've seen "true" and "false" written as if they were variables, but they're actually values. A variable which holds a true / false value is called a **boolean**, and it's defined using the keyword "bool" like so:

```
void main() {
  String text = "Hello World!";

  bool show = true;
  if(show) {
    print(text);
  }
}
```

Output: Hello World!

Now, the computer is checking if the variable's value is true. If, instead, you wanted to check is the condition was not true, you would use the **not symbol**: "!", like so:

```
void main() {
  String text = "Hello World!";

  bool show = true;
  if(!show) {
    print(text);
  }
}
```

No Output

Now, the computer is checking if the condition is not true, AKA checking if it's false. Thus, we have dead code again, and nothing is printed.

You can also clarify a set of code to run if, instead, the condition is false. By writing "else" right after the curly braces of the conditional close and specifying additional code within new curly braces, you can define **alternative code** for a conditional. Example:

```
void main() {
  String text = "Hello World!";

  bool show = true;
  if(!show) {
    print(text);
  } else {
    print(show);
  }
}
```

Output: true

The code successfully prints here because the computer is checking if *show* is not not true, AKA if it's true.

As a result, the value of *show* is printed. Oh yeah, Booleans can also be printed too.

Much like numbers, Booleans can also be defined beyond simply stating the direct value. One way to do this is through the **comparison operator** "==", which checks if two values are equal to each other.

```
void main() {
  String text = "Hello World!";

  bool show = true;
  if(2 == 2) {
    print(text);
  } else {
    print(show);
  }
}
```

Output: Hello World!

2 does indeed equal 2, so *text* is printed here. Instead, if we were to try:

```
void main() {
  String text = "Hello World!";

  bool show = true;
  if(2 == 4) {
    print(text);
  } else {
    print(show);
  }
}
```

Output: true

2 does not equal 4, so the alternative code is run, printing *show*.

Now that you've been shown functions, variables, operators, and conditionals, it's time to try making something useful!

Pythagorean Theorem Calc (Short for Calculator)

You should know what the Pythagorean theorem is, but just in case you don't:

The Pythagorean theorem states that the values of the two sides touching the 90-degree angle of a right triangle squares and added together is equal to the value of the diagonal of the triangle squared.

You can see how it would be helpful to know this, and even more helpful to have a program to work through this for you. So, let's get on with that!

Let's start first by defining the two input values, the two sides of the triangle, as integers using the "int" keyword:

```
void main() {
  int a = 3;
  int b = 4;

}
```

No Output

Now, let's print the two values squared and added together to get the third side squared:

```
1 void main() {
2   int a = 3;
3   int b = 4;
4
5   print(a * a + b * b);
6 }
```

Output: 25

Now we have the value of the third side, C, squared—but that's not what we want! To get the value of C, we need to take the square root of the value we have now.

There's no basic operator for calculating the square root, but there is a function! Functions also have the ability to provide values, including numbers. The name of the function is "sqrt", but when we try this:

```
1 void main() {
2   int a = 3;
3   int b = 4;
4
5   print(sqrt(a * a + b * b));
6 }
```

ERROR

...the computer yells at us. That's because it doesn't know what "sqrt" is! Don't worry, we don't have to write it ourselves... it already exists. All we have to do is import it.

You can add **imports** to a file by specifying the name of a library to import at the top of the file. In this case, you'd write the following:

```
1 import "dart:math";
2
3 void main() {
4   int a = 3;
5   int b = 4;
6
7   print(sqrt(a * a + b * b));
8 }
```

Output: 5

And now it knows what “sqrt” is, and it works great! Technically, the program works as intended now, but it’s a bit overcrowded.

One way we can clean this up a bit is by defining a new function. Let’s define a new function, just like we did for”main”, called “printC”:

```dart
import "dart:math";

void main() {
  int a = 3;
  int b = 4;

  print(sqrt(a * a + b * b));
}

void printC(){

}
```

Output: 5

Now we have an empty function. Go ahead and copy the print function over to this new function:

```dart
import "dart:math";

void main() {
  int a = 3;
  int b = 4;
}

void printC(){
  print(sqrt(a * a + b * b));
}
```

ERROR

...and it doesn’t work.

That's because the variables *a* and *b* are only defined for the function "main"—they're **local variables**. It can often get confusing when trying to understand where variables are defined, but try thinking of it this way: variables cannot escape the curly braces they're defined in. They have the ability to go into other curly braces, and they can escape the ones they enter, but they cannot leave the ones they were defined in. Also, a variable must be defined in a line before the line where it's referenced.

Understanding this, we can fix this issue either by moving the declaration of the variables to outside the "main" function, or we can move it into the "printC" function:

```
1 import "dart:math";
2
3 void main() {
4 }
5
6 void printC(){
7   int a = 3;
8   int b = 4;
9   print(sqrt(a * a + b * b));
10 }
```

No Output

This still doesn't do anything, though. Remember that the computer only looks for the "main" function when running a program, so it doesn't care about "printC" yet. This can be fixed by calling "printC" in "main":

```
1 import "dart:math";
2
3 void main() {
4   printC();
5 }
6
7 void printC(){
8   int a = 3;
9   int b = 4;
10  print(sqrt(a * a + b * b));
11 }
```

Output: 5

Now we're cooking! Except, we still have the same problem: the code is too complicated, and all we've done to fix it is move everything to a separate function. It would be better if our variables were defined within the function "main" but passed on to "printC".

Recall the print function, and how we were able to input variables into it. Variables inputted into a function are called **parameters**, and we can have more than one per function. Parameters are specified within the parentheses used to call a function and separated by commas, like so:

```
1 import "dart:math";
2
3 void main() {
4   int a = 3;
5   int b = 4;
6
7   printC(a, b);
8 }
9
10 void printC(){
11   print(sqrt(a * a + b * b));
12 }
```

Error

Why doesn't this work? Well, we've specified the variables to be input into the function, but we haven't specified that the function accepts parameters. To do so, we simply define the variables as parameters within the parentheses used to define the function, like so:

```
1 import "dart:math";
2
3 void main() {
4   int a = 3;
5   int b = 4;
6
7   printC(a, b);
8 }
9
10 void printC(int a, int b){
11   print(sqrt(a * a + b * b));
12 }
```

Output: 5

And now it works as intended!

Again, this program works fine as is, but it could be made better. What if we wanted to read the value of C somewhere else? That would require copying the square root function again, or would it?

Another thing that functions are capable of doing is providing **return values** (just like "sqrt"). So far, we've been defining functions using the keyword "void", which means that no value is returned (AKA void is returned). If, instead, we were to use a variable type, like "int", we could then return a value through the function using "return [vale];", like this:

```
1  import "dart:math";
2
3  void main() {
4    int a = 3;
5    int b = 4;
6
7    print(getC(a, b));
8  }
9
10 int getC(int a, int b){
11   return sqrt(a * a + b * b);
12 }
```

Error

There's no issue with our function here, it's just that we're attempting to return a double where an integer is required. We could change the type of the function to be a double, or we could **cast** the value into an integer—that is, we can change the type of the value. In Dart, casting values is done by:

```dart
import "dart:math";

void main() {
  int a = 3;
  int b = 4;

  print(getC(a, b));
}

int getC(int a, int b){
  return sqrt(a * a + b * b) as int;
}
```

Output: 5

Now, the double is converted into an integer, and the program works properly.

But what exactly is 5? We know that it's the value of C, but somebody else wouldn't. We need to make the printed value more user friendly. Let's start by converting our number into a String. This could be done by writing "as String", or we can use the more adaptable approach: **String interpolation**. By writing "${[variable]}" within a String, we can include the values of variables within a String, like this:

```dart
import "dart:math";

void main() {
  int a = 3;
  int b = 4;

  print("${getC(a, b)}");
}

int getC(int a, int b){
  return sqrt(a * a + b * b) as int;
}
```

Output: 5

We still have the same output, but if we were to write something else in out String:

```dart
import "dart:math";

void main() {
  int a = 3;
  int b = 4;

  print("c = ${getC(a, b)}");
}

int getC(int a, int b){
  return sqrt(a * a + b * b) as int;
}
```

Output: c = 5

Now it's looking better. This looks a little busy, so let's try defining our text String elsewhere, and then merging them through **String concatenation**:

```dart
import "dart:math";

void main() {
  String text = "c = ";

  int a = 3;
  int b = 4;

  print(text + "${getC(a, b)}");
}

int getC(int a, int b){
  return sqrt(a * a + b * b) as int;
}
```

Output: c = 5

This works, but String interpolation is generally preferred over String concatenation, so let's try:

```dart
import "dart:math";

void main() {
  String text = "c = ";

  int a = 3;
  int b = 4;

  print("$text${getC(a, b)}");
}

int getC(int a, int b){
  return sqrt(a * a + b * b) as int;
}
```

Output: c = 5

And it works just the same, but it looks better and is more professional. One last change to make for clarity: let's make our text variable final. **Final variables** are variables which cannot be altered after definition, making the program run a tiny bit better and making their purpose clearer. Since we never change the value of *text*, we may as well make it final:

```dart
import "dart:math";

void main() {
  final String text = "c = ";

  int a = 3;
  int b = 4;

  print("$text${getC(a, b)}");
}

int getC(int a, int b){
  return sqrt(a * a + b * b) as int;
}
```

Output: c = 5

...and it works just the same. If you want, you could also make *a* and *b* final as well.

Our program is done! But, will you remember how everything works? In the field of computer science, it can get very difficult to remember exactly how all of your code works, especially when working with others. That's why we use **comments**.

Comments are lines of code which don't effect how the program runs—they're only there to describe how it works to viewers. Comments can be started by "//" or sandwiched between "/*" and "*/", and can contain whatever text you want. For our final step, let's comment our code for clarity:

```dart
// Imports square root function
import "dart:math";

/*
 Main function.
*/
void main() {
  // Display text
  final String text = "c = ";

  // Input values
  int a = 3;
  int b = 4;

  // Prints the result
  print("$text${getC(a, b)}");
}

// Calculates c
int getC(int a, int b){
  // returns the square root of a squares + b squared
  return sqrt(a * a + b * b) as int;
}
```

Output: c = 5

And we're done! Good job!