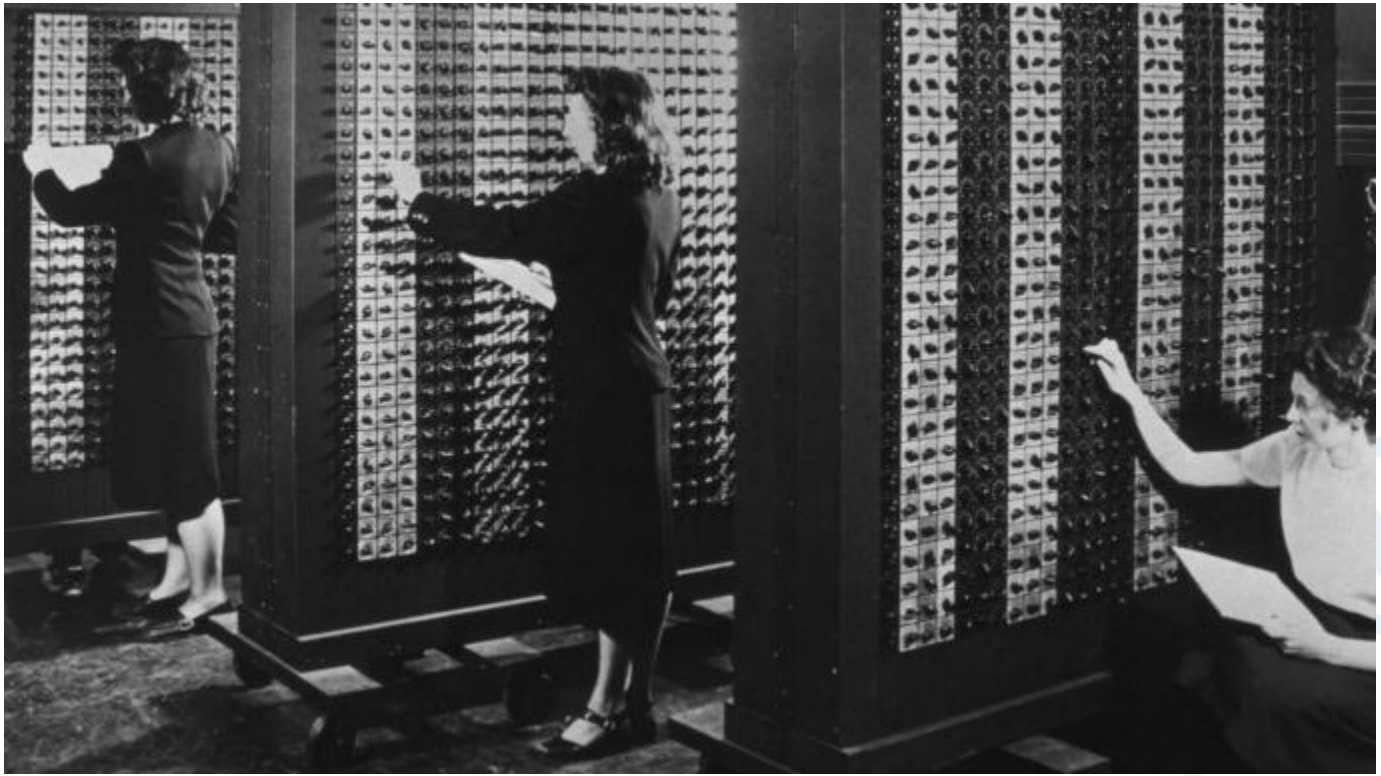
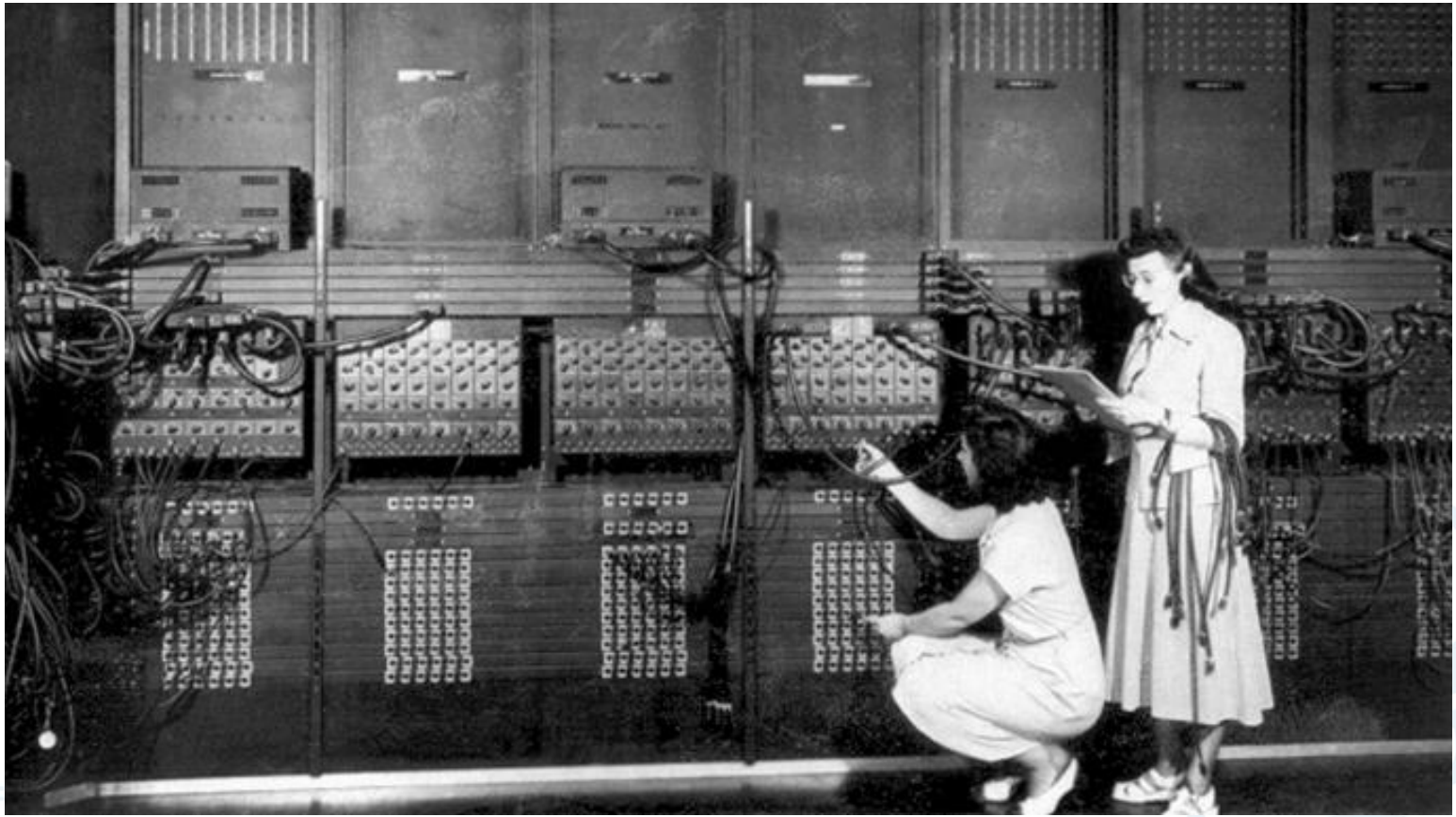
Programming Basics

Vivify Academy

VIVIFY
ACADEMY

Programming





Programming is the act of creating programs - sets of instructions - that are to be interpreted and executed by a computer (this includes phones and many other devices).

A programming language is a special language programmers use to develop software programs, scripts, or other sets of instructions for computers to execute.

Data

Programs operate on data - numbers, letters, binary streams etc. The goal of every useful program is to generate, manipulate, store and/or transfer data. Data in a program is thought of in terms of **values** - pieces of data of some type, that we can then use in a way that's useful to us. Examples:

- Visualization software
- Web browsers
- Spreadsheet software (e.g. Excel)
- Duplicate file finders

Examples of values

5

2.2

"Hello!"

true

simple values (literals)

5 + 2

10 / 2.5

2 > 8

(7 - 3) * (7 + 3)

expressions

Operators

Expressions are formed using values (which may be expressions themselves) and **operators**, such as +, -, *, /… Values used in expressions are called **operands**. There are different types of operators; the ones we used are **arithmetic** and **comparison** operators. There are also some other types, such as **logical, assignment, conditional** operators etc. We'll deal with each operator as we need it.

Data types

Computers don’t care about the type of data (because they can’t); but humans do, and programming languages were made for humans. Humans generally don’t perform the same operations on letters and numbers - you wouldn’t try to multiply 5 by “A”.

Programs that don’t make it obvious which data type is used where can be very difficult to understand.

Some simple types

- Numbers, such as 1, 2, -3, 10, 5.5
  - Whole numbers, or integers: 1, 2, -100
  - Real numbers, or “floats”, or “doubles”: 1.2, 3.125 (using decimal point, not comma!)
- Strings - sequences of characters, such as "Hello", "PHP", "a"
  - Single letters are also strings
  - Strings can be empty: ""
  - They can also be written with single quotes: 'This is a string too!'
- Logical, or Boolean values - **true** and **false** are the only possible values

Computers are stupid

Or, more accurately, they are not smart in the way humans are

- Their syntax is very strict, and they won’t tolerate some seemingly minor changes
- They won’t understand what you meant
- They work according to their own rules, which you may not understand yet (but you will)
- They are very literal: the string "a" is not the same as "A" (computers have no idea what letters are)
- If you “forget” a letter or make a typographical error (“typo”), they won’t correct you but will only tell you something is wrong.
- If you accidentally use the wrong word, operation etc., they may work (don’t crash) but you get the wrong output.
- ...

How we deal with computer stupidity - debugging

- With time, we learn to recognize different mistakes we may make, and know where to look to fix the program
- We control the program, so we can change (like stop it early) it to see exactly where the error occurs
- If the program is small enough, we can “execute” it ourselves, on a piece of paper, to see what happens and at which point the program starts acting in a way we don’t expect
- If an error occurs, the language will show you where it occurred, so you’ll have at least some idea where to start looking
- If all else fails, we can google the error, or ask someone more experienced

Running programs

Linux/Mac:

Create a file called “index.php”. In that folder, run “php -S localhost:8080”. To the top of the file, add “<?php”. After that, write anything you want.

In the browser, go to the address “localhost:8080”

Windows:

Install WAMP or XAMPP server, and find the folder that’s been set as the “www” folder (ask for help if needed). In that folder, edit the “index.php” file.

Did it work?

Unless it’s valid PHP code, it showed an error. Let’s see what the error says.

Trying something else

Type the following:

```
echo 1;
```

and reload the page. Type it exactly as you see it, with a semicolon at the end.

What happened?

The program worked! The command we wrote caused the number **1** to be written in the output box.

echo is a special command in PHP. It takes whatever comes after it, and writes it out onto the page. In this case, it was the number 1, but it can be anything. As for the semicolon, it has to be written at the end of each command. It's a signal to the programming language, telling it that the command is over and it can be executed. Programs are in most cases executed line-by-line, so it's important to "tell" the programming language when a particular statement can be executed.

Let’s try some other things too

Try executing this program:

```
echo 2.5;
```

Now try this:

```
echo 2.500;
```

```
echo 2 + 3;
```

Try these two commands, together:

```
echo 1;

echo 2;
```

What happens?

Let’s break some things!

Try executing these commands, one at a time:

```
echo 1

rcho 2;

echo Hello;

echo "Hello;

echo 2 / 0;

2 + 3;

"What happened?"

echo ;
```

Variables

While a program is running, it can store relevant data in variables. Variables are “containers” for data, and any type of data can be stored in a variable. Once some value is assigned to a variable, we can use that variable in any place where we need a value; the current value of the variable will “replace” it.

Syntax

Variables are created when you assign value to them for the first time - you don’t need any special command to create a variable. All variables must begin with a dollar sign ($). Value is assigned to a variable with the assignment operator (=).

```
$currentTime = "21:30";

echo $currentTime;
```

```
echo "21:30";
```

What is the point of variables?

Variables store the state of your program at every point. They are necessary when the program doesn’t have values ready, that is, when the values need to be inserted into the program from outside (for example from the keyboard, file, internet…). Also, they can be used to avoid doing some computations multiple times, especially if they are complicated.

```
$timeInHours = 20;

$timeInMinutes = $timeInHours * 60;

$timeInSeconds = $timeInMinutes * 60;
```

The assignment operator

The assignment operator may be confusing at first; even though the “equals” (=) sign is used, this operator doesn’t represent equality. For an equality check, there are two other operators, double-equals (==) and triple-equals (===). This is often difficult to get used to, and many beginning programmers make mistakes, using the assignment operator when they mean equality. It’s usually not a big problem, but it’s one more thing to keep in mind.

Some more expressions

```
$a = 1;
$b = 4;
echo $a + $b;

$c = 3;
$d = 3;
echo $c === $d;
```

```
$dayOfWeek = "Tuesday";
echo "Today is ";
echo $dayOfWeek;

$w = "String 1";
$x = "String 2";
$y = "String 2";
$z = "string 1";
var_dump($x === $y);
var_dump($w === $z);
```

Logic operators

PHP (and most other languages) has a few operators meant to be used with boolean values.

- AND operator (&&), which is true only if both operand
- OR operator (||), which is true if either operand is true (or both)
- NOT operator (!), which is the opposite of its operand

PHP allows operands of any type to be used in place of booleans, and evaluates them according to certain rules. For now we will be working with booleans only.

&&

The “and” operator. Result is true only if both operands are true, otherwise it’s false:

- **(true && true)** is true
- **(true && false)** is false
- **(false && true)** is false
- **(false && false)** is false

Used when multiple conditions must be satisfied:

$canPlayOutside = $doneHomework && $notRaining;

||

The “or” operator. Result is true if either or both of its operands are true, otherwise it’s false. It can also be stated as: result is false only if both operands are false, otherwise it’s true:

- **(true || true)** is true
- **(true || false)** is true
- **(false || true)** is true
- **(false || false)** is false

Used when any condition is enough:

```
$mustStayInside = $hasNotDoneHomework  ||  $raining;
```

!

The “not” operator. Result is true if the operand is false, otherwise it’s true:

- **!false** is true
- **!true** is false

Used to invert a condition:

$canPlayOutside = !$mustStayInside;

$canPlayOutside = $doneHomework && !$raining;