## Essential Objects

<table>
<thead>
<tr>
<th>Class</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Part</td>
<td>A physical brick in the world.</td>
</tr>
<tr>
<td>Model</td>
<td>A container for Parts.</td>
</tr>
<tr>
<td>Folder</td>
<td>A container for Scripts and value objects.</td>
</tr>
<tr>
<td>Script</td>
<td>A container for Lua source code.</td>
</tr>
<tr>
<td>LocalScript</td>
<td>A Script that runs its code on a client.</td>
</tr>
</tbody>
</table>

## Basic math functions

<table>
<thead>
<tr>
<th>Operation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>a + b</td>
<td>Adds a and b.</td>
</tr>
<tr>
<td>a - b</td>
<td>Subtract a and b.</td>
</tr>
<tr>
<td>a * b</td>
<td>Multiply a and b.</td>
</tr>
<tr>
<td>a / b</td>
<td>Divides a by b.</td>
</tr>
<tr>
<td>a % b</td>
<td>Remainder of a divided by b.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>math.random(n)</td>
<td>Returns random number from 1 to n (no negatives).</td>
</tr>
<tr>
<td>math.random(a, b)</td>
<td>Returns random number from a to b.</td>
</tr>
<tr>
<td>math.max(...)</td>
<td>Returns the largest number.</td>
</tr>
<tr>
<td>math.min(...)</td>
<td>Returns the smallest number.</td>
</tr>
</tbody>
</table>

## Basic math functions (cont)

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>math.floor(n)</td>
<td>Rounds n down.</td>
</tr>
<tr>
<td>math.ceil(n)</td>
<td>Rounds n up.</td>
</tr>
<tr>
<td>math.abs(n)</td>
<td>Returns absolute value of n.</td>
</tr>
<tr>
<td>math.sqrt(n)</td>
<td>Returns square root of n.</td>
</tr>
<tr>
<td>math.pi</td>
<td>Approx equal to 3.14159</td>
</tr>
</tbody>
</table>

It's important to work out problems by hand before translating their solutions into code. Algebra is necessary for success. Read about all math functions here.

## String functions

<table>
<thead>
<tr>
<th>Operation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>a .. b</td>
<td>Combine two strings.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>string.len(str)</td>
<td>Returns length of str.</td>
</tr>
<tr>
<td>string.upper(str)</td>
<td>Returns str in upper-case.</td>
</tr>
<tr>
<td>string.lower(str)</td>
<td>Returns str in lower-case.</td>
</tr>
<tr>
<td>string.reverse(str)</td>
<td>Returns str in reverse.</td>
</tr>
<tr>
<td>string.rep(str, n)</td>
<td>Returns str repeated n times</td>
</tr>
</tbody>
</table>

A string is a collection of characters, or text. An example of a string property is the `Name` property. Read all string manipulation functions here.

## Tables

```lua
local list = {1, 2, 3}
local firstNum = list[1]
list[2] = 4
print("There are " .. #list .. " numbers")
local total = 0
for i = 1, #list do
    total = total + list[i]
end
print("The total is " .. total)
```

Tables are a collection of values. They are defined using curly braces {} with values separated by commas. Access the values inside using square brackets []. Tables are sometimes called arrays. Use a `for` loop to work with all items in a table individually. The :GetChildren() method returns a table of children in an object.
### Creating objects

**Finding Objects**

- `workspace.Parent:Destroy()`
- `print(script.Parent.Name)`
- `game.ServerStorage.Tree:Clone()`

Use a period to access an object's children. Use `.Parent` to access an object's parent. Use constants like `game`, `workspace`, and `script` to identify objects in the hierarchy.

**Creating objects**

**How do I create an object?**

Using `Instance.new` and setting the parent:

```lua
object.Parent = parent
```

**How do I access an object's properties?**

Use a period (`.`):

```lua
print(object.Name)
```

**How do I set an object's properties?**

Use a period (`.`) and equals sign (`=`):

```lua
part.Transparency = .5
```

**How do I destroy an object?**

Using `object:Destroy()`

### Constants

- `game`: Parent of all game services.
- `workspace`: Container for all bricks and models are stored.
- `script`: The currently running script.

### General Object Functions

<table>
<thead>
<tr>
<th>Method name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>FindFirstChild(name)</code></td>
<td>Return a child with name or <code>nil</code> if it doesn't exist.</td>
</tr>
<tr>
<td><code>WaitForChild(name)</code></td>
<td>Pauses until a child with a name exists and returns it.</td>
</tr>
<tr>
<td><code>IsA(className)</code></td>
<td>Return whether the object is a certain type of object.</td>
</tr>
<tr>
<td><code>Clone()</code></td>
<td>Makes and returns a copy of an object.</td>
</tr>
<tr>
<td><code>Destroy()</code></td>
<td>Permanently delete an object.</td>
</tr>
<tr>
<td><code>GetChildren()</code></td>
<td>Return a list of an object’s children.</td>
</tr>
</tbody>
</table>

These are functions (aka methods) for all classes of ROBLOX objects. Read about all methods here.

### Event basics

**Event basics**

- `function onTouch(part)`
  - `print(part.Name .. " touched me!")`

Events are specific occurrences relating to objects. When an event `fires`, or occurs, all connected functions are called.

### Basic functions

- `wait(n)`: Wait `n` seconds then continue.
- `print(...)` Display something in the Output window.

### Variables

- `local myScore = 5`
- `myScore = myScore + 1`
- `print(myScore)`
- `local myName = "Ozzypig"`
- `print("My name is " .. myName)`

Variables store data of any kind - numbers, strings, tables, objects or `nil` (nothing). A `local` variable is only accessible in the block of code it is defined in.
### If statements

```lua
if workspace:FindFirstChild("Tree") then
    print("There is a tree here.")
end
if coins < 5 then
    print("You need more money.")
else
    print("You have enough money!")
end
if player.Name == "Jake" then
    print("You are an awesome guy, Jake")
elseif player.Name == "Sally" then
    print("You are a sweetheart, Sally")
else
    print("You are a pretty cool person")
end
```

If statements will run their code if the value between `if`/`then` is true (or not nil). They can one an `else` block, or any number of `elseif` blocks.

### Loops

#### Numeric for loop

For counting numerically.

Example: Count from 1 to 5:

```lua
for i = 1, 5 do
    print(i)
end
```

#### Generic for loop

Most often used for object children.

Example: Print all children in object:

```lua
for i, child in pairs(object:GetChildren()) do
    print(child.Name)
end
```

#### While loop

Perform code until a condition is false.

Example: Remove all children named 'Ball'

```lua
while object:FindFirstChild("Ball") do
    object.Ball:Destroy()
end
```

#### Repeat-until loop

Perform code once, then again until a condition is true.

Ex.: Copy objects until there are 5.

```lua
repeat
    newObject = object:Clone()
    newObject.Parent = workspace
    wait(1)
until #workspace:GetChildren() >= 5
```

Loops are used to **iterate**, or repeat code a number of times.

### Function examples

```lua
function sayHello()
    print("Hello, world")
end
sayHello()

function addTwoNumbers(a, b)
    print("The sum is:", a + b)
end
addTwoNumbers(3, 5)

function calculateSquare(n)
    return n * n
end
local result = calculateSquare(3)
```

A function is a named block of code that can be run anywhere in code by calling it by name. Functions can have arguments (given values) and/or return values.

---

By Ozzypig (Ozzypig)
cheatography.com/ozzypig/ozzypig.com

Published 25th January, 2016.
Page 3 of 3.

Sponsored by CrosswordCheats.com
Learn to solve cryptic crosswords!
http://crosswordcheats.com