
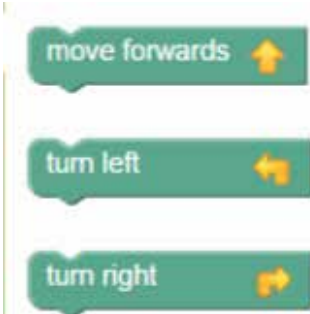


Blockly – Python Phrasebook



Blockly	Python	Notes
 <p>Van starting block</p>	<pre>import van v = van.Van()</pre>	<p>The starting code has been created 'behind the scenes' in Python and has to be imported.</p> <p>Those two lines are therefore automatically added to your new program in Rapid Router. When you open the Python pane, you should see this code. These two lines are required before all other commands.</p> <p>After this code, the van object, which is the object controlling van movement on the map, is indicated by the variable v.</p> <p>You could change the name of this object to whatever you like, but be consistent and use the same name with the commands afterwards.</p>
 <p>Move forwards Turn left Turn right</p>	<pre>v.move_forwards() v.turn_left() v.turn_right()</pre>	<p>move_forwards, turn_left and turn_right are, in fact, procedures. Once called, they cause the van to move. This is why when we want the van to move forwards, we call its move_forwards procedure by putting () after its name (e.g. "v.move_forwards()")</p> <p>Note the underscore ("_") is an essential part of the name.</p>



Repeat n times

```
for number in range(n):
```

```
for count in range(3):
    v.move_forwards()
```

This code will repeat all commands indented underneath it n times, where n is a whole number such as 3. It works with any name inserted where the word number is shown in the example. (The built-in range function just tells the loop how many times to repeat.)

The example on the left would repeat `v.move_forwards()` 3 times. You can use the count variable inside the body of the loop if you want the van's behaviour to differ depending on how many times the for loop was executed. On the first time through the loop, it's value will be 0, incremented by one each time around the loop.

You must use a colon `:` at the end of the for line because this indicates that a sequence of instructions should follow (we call them the body of the loop). These instructions must all be indented by the same amount (ideally by 4 spaces).



wait

```
v.wait()
```

A procedure which when executed or 'called', causes the van to wait.



if, else if and else

```
if v.is_road('FORWARD'):
    v.move_forwards()
elif v.is_road('LEFT'):
    v.turn_left()
else:
    v.turn_right()
```

The indentation of the instructions inside a block must match. Ideally, they would be indented 4 spaces more than the previous statement.

`if`, `elif` and `else` statements must all be indented to the same level as each other, and all require a colon `:` at the end of their statement.

(note: `elif` is short for 'else if')

The example to the left will cause the van to wait if it is currently at a red traffic light.



```
if v.at_traffic_light('RED'):
    v.wait()
```

Conditions:



```
v.is_road('FORWARD')
v.is_road('LEFT')
v.is_road('RIGHT')
v.at_traffic_light('RED')
v.at_traffic_light('GREEN')
```

These conditions can be added after an `if`, `elif` or `else` statement.

Each of the conditions are functions that check the state the van is in (i.e. what kind of road is there ahead or what colour the traffic light is) and returns `True` or `False`.

If a variable can only be `True` or `False`, it is called a “Boolean”.

Repeat until at destination
or
Repeat while not at destination



```
while not v.at_destination():
    v.move_forwards()
```

The `while not` statement repeats until the condition is `True`.

Remember the colon denotes a set of instructions to be followed if the `while` condition is met (we call it the body of the loop).


These instructions must be consistently indented, ideally by 4 spaces.

This example will cause the van to wait until the traffic lights are no longer red.

Repeat while traffic lights red



```
while v.at_traffic_light('RED'):
    v.wait()
```

<p>Procedures:</p> 	<pre>def procedurename(): def proc1(): v.turn_left() v.turn_right() proc1() def forward_left(n): for count in range(n): v.move_forwards() v.turn_left() forward_left(4)</pre>	<p>To create a procedure, you use the <code>def</code> keyword. The procedure needs a meaningful name where <code>procedurename</code> is placed in the example. You must have a pair of brackets <code>()</code> and a colon <code>:</code> after it.</p> <p>All subsequent statements that are to be part of the procedure must be indented to the same level as each other (ideally 4 spaces).</p> <p>The procedure is then executed (or called), by typing the name of the procedure followed by a pair of brackets <code>()</code>.</p> <p>In this example, the <code>proc1</code> procedure will move the van left and then right when called. In reality, it is better to choose a more meaningful name for your procedure.</p> <p>What happens inside the procedure can be changed each time by passing in arguments. Arguments available to the procedure are defined in between the two brackets, such as the argument <code>n</code> in this example. This value is then used to change how many times the loop is executed.</p> <p>When calling a procedure with an argument, you must define the value for that argument when you call the procedure. In this case, we are calling the <code>forward_left</code> procedure with the argument value of 4, which means the loop will execute 4 times (the van will move forward 4 times before turning left).</p>
<p>Outputting text No equivalent in Blockly</p>	<pre>print ('message')</pre>	<p>This will display whatever you put in the place of <code>message</code> in the Rapid Router Output console. You should surround your message with single quotes.</p>
<p>Variables No equivalent in Blockly</p>	<pre>var_1 = 'Hello!' var_2 = var_1 n = 1</pre>	<p>You can create your own variables using any name except those on the reserved list below. You can assign values to variables using the <code>=</code> operator. You can assign the value of one variable to another variable.</p> <p>You can use the variables anywhere in your program that you would normally use whatever the variable stands for.</p>

Mathematical operators No equivalent in Blockly	<pre>n = 2 + 3 n = 2 - 3 n = 2 * 3 n = 2 / 3</pre>	<p>You can use these with numbers or numeric variables or both.</p> <p>The examples show addition, subtraction, multiplication and division.</p> <p>With division, note that the result will be rounded down. For example, the following:</p> <pre>n = 5 / 2</pre> <p>The value of n will be 2, not 2.5.</p>
Exit a loop early No equivalent in Blockly	<pre>while condition: break</pre>	<p>Instead of waiting until <code>condition</code> is no longer <code>True</code>, the <code>break</code> statement will stop executing the loop immediately.</p>
Skip to the next time round the loop No equivalent in Blockly	<pre>for count in range(5): continue</pre>	<p>Instead of executing the rest of the commands in the loop, this statement will immediately move onto the next time around the loop.</p>
Do nothing No equivalent in Blockly	<pre>pass</pre>	<p>This statement does absolutely nothing. This is required if you want to define a procedure (or another block such as <code>for</code> or <code>while</code>) that has no other commands.</p>

Reserved words ie words that cannot be used as variable or procedure names (see below):

and	exec	not	asset	finally	or	break	for
pass	class	from	print	continue	global	raise	def
if	return	del	import	try	elif	in	while
else	is	with	except				