PYTHON REPL

DESCRIPTION

The original computer interface.
RECEIVES text from keyboard
EVALUATES text as instructions
PRINTS the result
LOOPS back to receiving instructions.
Hence R E P L

CONNECTING TO REPL

From a Terminal on a laptop
Type ***vanguard shell*** and press enter
Press ***CTRL+C*** to get a **>>>** prompt

```
Launching Miniterm to connect to Vanguard Python
--- Miniterm on /dev/ttyUSB0  115200,8,N,1 ----
-- Quit: Ctrl+] | Menu: Ctrl+T

>>>
```

```
4+4
8
>>>
```

Type ***4+4*** at the prompt, press *Enter*. Evaluating this expression results in the value 8. After printing ***8*** the REPL loops back to showing the **>>>** prompt

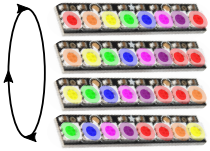
```
rainbow()
>>>
```

Type ***rainbow()*** at the prompt, press *Enter*
The display shows a spectrum of colours.
The prompt is ready for another instruction

```
rotateSpectrum()
```

```
>>>
```

Type ***rotateSpectrum()*** and press *Enter* to trigger an animated rainbow. It won't prompt for another instruction while it is busy.



Press ***CTRL+C*** to stop the animation and get back to the prompt

vgkits

www.vgkits.co.uk/rainbow

COMMANDS

DESCRIPTION

Like sending pre-agreed instructions to the engine room of a ship.

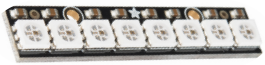
ISSUE A COMMAND

A **word** then round brackets **()**
Pass any **values** in the brackets
Multiple values need a **comma**

```
rainbow()
clearPixels()
sleep(2)
setPixel(0, red)
```

`clearPixels()`

Set all lights to black (off)

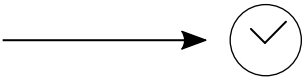


`setPixel(0, red)`

Set the first light to red

`sleep(2)`

Do nothing for two seconds



`rainbow()`

Show a color spectrum

VGkits www.vgkits.co.uk/rainbow

NAMING

DESCRIPTION

Like putting a value or a command in a labelled drawer to retrieve later.

NAMING VALUES

A **name** then **=** followed by the **value** to store.
Later, use the word in place of the value

```
intro = "Why was 6 afraid of 7?"
numbers = list(range(7, 10))
punchline = "Because " + str(numbers)
print(intro)
print(punchline)
```

```
favouriteColor = red
favouriteColor = blue
favouriteColor = purple
```

Named values are called variables.
Their values can be varied, and commands using the names then see a changed value

NAMING ROUTINES

def before a **name** then brackets **()** and a colon **:**
Commands indented with **tabs** one-per-line after the colon.
Empty line ends the routine.
See BLOCKS for indentation
See COMMANDS for triggering

```
def tellJoke():
    print("Why was 6 afraid of 7?")
    numbers = list(range(7, 10))
    print("Because " + str(numbers))

tellJoke()
```

```
def setAll(color):
    for pos in range(0, 8):
        setPixel(pos, color)
```

Define setAll as shown, then set all lights blue using

```
setAll(blue)
```

vgkits www.vgkits.co.uk/rainbow

COLOURS

DESCRIPTION

Each pixel in your display actually has 3 Light-Emitting Diodes (LEDs) inside.
The Colours in our programs are lists of numbers. These describe the brightness of the Red, Green and Blue LEDs.

MAKING COLOUR VALUES

There are many ways to generate colours for the setPixel() command. We can create a list of three numbers less than 256, use the named lists already available or use colour functions to calculate lists for us.
Try varying these numbers. What happens?

```
setPixel(0, yellow)
setPixel(1, [255,255,0])
setPixel(2, hueToRgb(0.1666667))
```

```
red
[255, 0, 0]
>>>
```

Type ***red*** at the prompt, press *Enter*. This named value is a list of numbers ***[255, 0, 0]***. After printing it, the REPL loops back to showing the **>>>** prompt.
Try other colours. What do you notice?

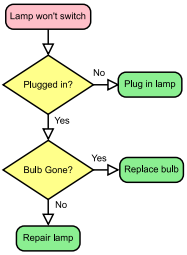
```
darkenRgb(red)
[31, 0, 0]
>>>
```

Type ***darkenRgb(red)*** and press *Enter*.
Note that capitalisation of darkenRgb matters.
Can you send darker colours to your display?

```
hueToRgb(0.5)
[0, 255, 255]
>>>
```

Type ***hueToRgb(0.5)*** and press *Enter*.
Note that capitalisation of hueToRgb matters.
Try changing the number from 0.5?
Can you send these colours to your display?

BLOCKS



DESCRIPTION

Like a decision in a flow chart, a block evaluates its commands only when a criterion is met.
Blocks and their commands are processed in sequence, one by one.

BLOCK STRUCTURE

A ***declaration*** followed by a colon **:**
Indented lines define block commands
Unindented empty line ends the block

```
if lemons in life:
    makeLemonade()
    sellLemonade()
```

```
if fits(shoe):
    wear(shoe)
```

An ***if*** block tests if a condition is true before evaluating its commands

```
while iron == hot:
    strike()
```

A ***while*** block evaluates its commands *repeatedly* as long as its condition is true

```
for cloud in clouds:
  extractSilver(cloud.lining)
```

A ***for*** block evaluates its commands once for each item in a list, until the list is exhausted or *break* is called

vgkits vgkits.co.uk/rainbow

REPEATING

DESCRIPTION

Like repeating the same steps to every item on a conveyor belt

'FOR' LOOPS

Type this example, using ***Enter*** to start a new line and ***Tab*** to indent lines. Enter an empty line (deleting any tabs) to finish the procedure.
This for loop repeats three indented steps, with pos given a new value each time, starting at 0 and stopping at 8.
What will it do when you run it?

```
for pos in range(0, 8):
    clearPixels()
    setPixel(pos, red)
    sleep(1)
```

'WHILE' LOOPS

This while loop checks if a condition is true then repeats the indented steps.

1==1 checks if 1 is the same value as 1. This will always be true so the steps will repeat forever.

What effect will these steps create?

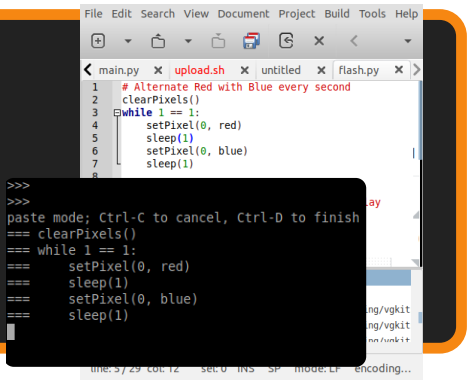
```
clearPixels()
while 1 == 1:
    setPixel(0, red)
    sleep(1)
    setPixel(0, blue)
    sleep(1)
```

USING A TEXT EDITOR

To make changes more easily, keep your code in a python-aware text editor like Geany. After you've made any changes you can copy-paste it into the REPL to run it.

PASTE MODE

First copy the text of your code with the mouse. Then begin paste mode in the REPL by pressing ***CTRL+E***. Paste your code with the mouse. Finally, end paste mode to run your script by pressing ***CTRL+D***.



vgkits vgkits.co.uk/rainbow

NESTING

DESCRIPTION

Like boxes nested inside other boxes. When a block is evaluated, commands and further blocks inside it get evaluated one by one.

NESTED BLOCKS

Each block is indented by one tab more than its containing block

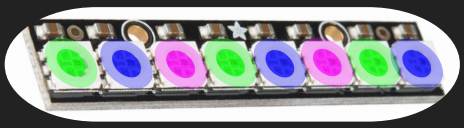
```
def clearPixels():
    for pos in range(0, 8):
        setPixel(pos, black)
```

EXAMPLE

The ***fancyColors*** routine fills the display with a repeating pattern

It has a def block
containing a while block
containing a for block
containing an if block
and an else block

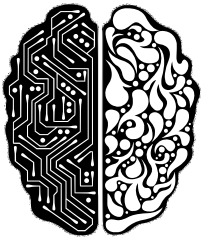
EXPLANATION
Beginning with first pixel (0)
while there are pixels left
for each color in the pattern
  set pixel at pos to the color
  add 1 to pos
  if pixels left, continue 'for loop'
  else break 'for loop'



```
def fancyColors():
    pos = 0
    while pos < 8:
        for color in [green, blue, purple]:
            setPixel(pos, color)
            pos = pos + 1
            if pos < 8:
                continue
            else:
                break
```

vgkits.co.uk/rainbow

ADVANCED



DESCRIPTION

Combine many features in one program.

BEHAVIOUR

This routine uses a library which can generate random numbers. It assigns random colors to random pixels every half second.

```
# Sets random pixels to random colors
from vgkits.random import randint
clearPixels()
colors = [red, blue, green, yellow, purple]
while 1 == 1:
    pixelPos = randint(8)
    colorPos = randint(len(colors))
    color = colors[colorPos]
    color = darkenRgb(color)
    setPixel(pixelPos, color)
    sleep(0.5)
```