Head First Programming

A Brain-Friendly Guide

A learner’s guide to programming using the Python language

Impress the boss with your debugging skills

Reduce, Reuse, and Recycle your code with functions and parameters

Use loops to help Hank get his girl

Use objects to control a fleet of robots

Keep everyone happy at lunchtime with the design-code-test cycle

Vern Ceder
Would you like to really take control of your computer?

Have you ever wished you could take control of your computer? Not just run some canned software written by others, but create your own programs? Well, you can! First you need to learn a few phrases in a language your computer understands, and then, once you’re speaking the same language, you can truly rule your machine.
It's art man, can you dig?

Pimpmycube.com has more work coming in than their resident artist can handle with his pair of compasses, protractor, and pens.

The big boss has seen your program, and with it the future, and it doesn't look like there's a lot of room for an artist who takes a few days to finish some of the more complex drawings the site offers...

Yeah, yeah, that's great, but it only takes me a few minutes to draw a pattern like that. And I can use a bunch of other colors. Black is SO last season...

Jacques, soon to be starving artist...

Using the color command and the list of colors below, change your program so that each of the squares will be a different color in the output.

<table>
<thead>
<tr>
<th>Command</th>
<th>What it does</th>
</tr>
</thead>
<tbody>
<tr>
<td>forward(pixels)</td>
<td>moves forward (distance to move)</td>
</tr>
<tr>
<td>backward(pixels)</td>
<td>moves backward (distance to move)</td>
</tr>
<tr>
<td>right(degrees)</td>
<td>turns right (angle to turn through)</td>
</tr>
<tr>
<td>left(degrees)</td>
<td>turns left (angle to turn through)</td>
</tr>
<tr>
<td>color('colorname')</td>
<td>draws with a color</td>
</tr>
</tbody>
</table>

forward(100)
right(90)
forward(100)
right(90)
forward(100)
right(90)
forward(100)
right(45)← Turtle turns to the right.
forward(100)
right(90)
forward(100)
right(90)
...

'aquamarine' 'blue' 'chocolate'
'gainsboro' 'gray' 'green' 'honeydew'
'ivory' 'lavender' 'magenta' 'maroon'
'navy' 'orange' 'orchid' 'pink'
'purple' 'red' 'sienna' 'thistle'
'tomato' 'violet' 'yellow'
Your job was to use the color command and the list of colors to change your program so that each of the squares would be a different color in the output.

Did you change the color of your first square before you started drawing it?

Changes the color of our second square.

Changes the color of our third square.

Changes the color of our fourth square.

Make sure your color names are in quotes!

If you get an error message in the shell, double check that your color names are in quotes. We’ll talk about why you need the quotes in chapter 2.
Change is painful

Your first colored drawings are a big hit. In fact, they’re almost too big a success—every one you run into has an idea for a variation that the big boss would like to try. And all of the variations seem pretty cool... to everyone else. To you making all of those changes in your code is getting old. One problem is that everyone seems to want different sizes, and it’s a real pain to change the distance traveled in every single forward(100) statement.

Repeating that much code means you’re taking longer to write that simple program than I am to draw the same thing!

Computers are good at doing the same thing over and over, right? You can save time by giving your computer a name to use in place of a value at the start of the program, and then repeating it.

Sharpen your pencil

Look at the code below, then write down what you think it’s doing.

```plaintext
distance = 100
forward(distance)
right(90)
forward(distance)
...
```

Code continues
Chapter 1

Change values in just one place

We can use this same “name-in-place-of-a-value” trick with almost anything—the distance, the degrees to turn, even the color names:

```python
main_color = "aquamarine"
distance = 100
turn = 120
color(main_color)
right(turn)
forward(distance)
right(turn)
...
```

The advantage of using a name for the value is that if you use the same value a lot, you can just change the value where you name it. You don’t need to mess with changing the actual value everywhere it appears in your code.

Say the big boss wants a bigger version of this drawing where the main color is sienna and the turn is larger. Easy. Now all we need to do, is change three values where we set them to a name, and we can leave the rest of our code. That’s just three changes instead of 24!

```python
main_color = "sienna"
distance = 200
turn(60)
color(main_color)
right(turn)
forward(distance)
right(turn)
...
```
Below is the code for one square of the pattern we drew above. Rewrite the code so that it uses names instead of values. Be sure to give values to the names.

```python
color('navy')
forward(100)
right(90)
forward(100)
right(90)
forward(100)
right(90)
forward(100)
right(45)
```
Your job was to rewrite the code so that it uses names instead of values, making sure to give values to the names.

```python
my_color = 'navy'
distance = 100
square_turn = 90
main_turn = 45

color(my_color)
forward(distance)
right(square_turn)
forward(distance)
right(square_turn)
forward(distance)
right(square_turn)

This gives a name to the color.
This sets 'distance' to the number of pixels.
This sets square_turn to be 90.
This sets main_turn to be 45.

The code uses the names we created, not the actual numbers.
```
Repetition is still a problem

Unfortunately, he's right, and if you want to make your designs more complex you'll be repeating even more code.

Just turning a square 8 times makes a fairly cool design, enough to see that you could really bury Jacques with designs repeating octagons, say, 20 times.

The problem is that drawing an octagon takes 8 sides instead of 4 and then repeating that 20 times would be 160 repetitions! Ack!

Can you think of ways that we might be able to repeat the same chunks of code over and over?
Code can repeat

Luckily for us (and not-so-luckily for Jacques) there is a way to make Python repeat chunks of code. We’ll talk about it more in a couple of chapters, but for now, we can just use the pattern below—a line with `for x in range(some_number):`
where `some_number` is the number of times you want to repeat, followed by what you want to repeat—in this case, drawing one side of a square.

```python
sides = 4
distance = 100
square_turn = 90
for side in range(sides):
    forward(distance)
    right(square_turn)
```

The beauty of programming is that you can build up from small parts to big parts, so if we have some code that makes a square, even if it’s done by repeating code, we can also repeat our repeated code to make something even more complex.

Before, we made a design by drawing a square 8 times, so let’s see if we can use the pattern for making a square above and repeat it 8 times.

```python
from turtle import *
tracer(False)
figures = 8
main_turn = 45
sides = 4
distance = 100
turn = 90
for x in range(figures):
    for side in range(sides):
        forward(distance)
        right(turn)
    right(main_turn)
```

Watch it!
The way that Python tells what gets repeated depends on the indenting so copy it carefully.
Making the figures more complex

To draw an octagon instead of a square, and then repeat it 20 times instead of 8 is a piece of cake. Since our code uses names instead of the actual values, the only things we have to change are the spots where we put values to the names.

```
figures = 8  # This needs to be the number of octagons we want.
main_turn = 45  # This needs to be changed for 20 turns.
sides = 4  # This needs to be 8 for an octagon.
distance = 100  # We'll leave distance the same.
turn = 90  # This needs to be changed for an octagon.
```

Have a shot at making that octagon design. Just take the pattern over on page 28 and adapt it by changing the values. We'll need to make an octagon (8 sides) repeat 20 times. Use whatever color you want.
Your job was to make the octagon design using the pattern on page 28 and adapting it by changing the values.

```python
from turtle import *
tracer(False)
figures = 20  # Sets figures to its new value.
main_turn = 18  # 360/20 is 18 degrees.
sides = 8  # 8 sides for an octagon.
distance = 100  # 360/8 is 45 degrees for each turn in an octagon.
turn = 45
color('violet')
for x in range(figures):
    for side in range(sides):
        forward(distance)
        right(turn)
        right(main_turn)
```

Your job was to make the octagon design using the pattern on page 28 and adapting it by changing the values.

Make it random

Like the man said, black is so last season. But so is sticking with just one predictable color. It would be much cooler if we could have changing, unpredictable colors. And we can! If we set the color to 3 random values, we’ll get a random color.

We need to import the “random” library (just like we do “turtle”) and then use `random()` in place of color values.

```python
from turtle import *
from random import *

...  

color(random(), random(), random())
```

Set your color to `random()` three times to get a random color.