

Hangman

One Project in Detail

In this section, we use Pencil Code to make a game of hangman from scratch.

It takes a couple hours to learn enough programming to make a game of hangman.

We will learn about:

- Memory and naming
- Computer arithmetic
- Using functions
- Simple graphics
- How to make a program
- Input and output
- Loops and choices
- Delays and synchronization
- Connecting to the internet

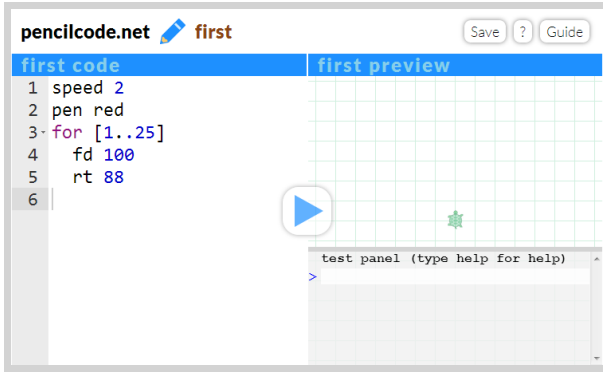
At the end we will have a game we can play.

1. Running Pencil Code

Go to pencilcode.net.

Click on "Let's Play!"

The screen should look like this:



The left side of the screen is where you type in your program, and the right is where programs run. The lower right corner is a test panel where you type code and run it right away.

While exploring the projects in this book, you can also use the test panel in the lower right corner to ask for help with how commands work.

```
test panel (type help for help)
> help
help is available for: bk cg cs ct fd ht if ln lt rt st abs cos dot
...
>
```

The characters that you should type will be highlighted.

2. Keeping a Secret

We will begin by working in the test panel.

CoffeeScript can remember things. Let's tell it a secret word.

Type the blue words below into the test panel.

```
test panel (type help for help)
> secret = 'crocodile'
```

See what happens when you press Enter.

```
test panel (type help for help)
> secret = 'crocodile'
"crocodile"
> █
```

Reveal your secret by typing "write secret".

```
> write secret
> █
```

Check the upper right panel!

Typing just the name in the test panel will reveal the word there.

```
> secret
"crocodile"
> █
```

Now try something CoffeeScript doesn't know. Try typing "number".

```
> number
▶ number is not defined
> █
```

Don't worry. This is fine. You just need to teach CoffeeScript what "number" is and try again.

```
> number = 43
43
> number
43
> █
```

3. Computers are Fine Calculators

A computer is better than any calculator at doing math. Let's try.

```
> 2+33+66
101
```

In CoffeeScript, plus and minus use the usual symbols + and -. Times and divide are done using the * and / symbol.

```
> 33333333 * 44444444
1481481451851852
```

Named values can be used in formulas.

```
> n=123456789
123456789
> n*n*n
1.8816763717891548e+24
```

The e+24 at the end is the way that large numbers are written in CoffeeScript. It means $1.8816763717891548 \times 10^{24}$. CoffeeScript calculates numbers with 15 digits of precision.

There are several ways to change a number. For example, += changes a variable by adding to it.

```
> n += 1
123456790
> n
123456790
> █
```

Some symbols to know:

code	meaning	code	meaning	code	meaning
+	plus	x = 95	save 95 as x	word.length	the length of word
-	minus	x is 24	is x equal to 24?	String(num)	turns num into a string of digits
*	times	x < 24	is x less than 24?	Number(digits)	makes a number from a string
/	divide	x > 24	is x more than 24?	n += 1	change n by adding one

These operations can be combined.

CoffeeScript obeys the same order of operations used in Algebra.

What will it do when we say "String(99 * 123).length"?

What will it say for (2 * 3 + 3 * 5) / 7 - 1?

Try your own fancy formulas. Don't worry if you get errors.

4. Strings and Numbers

What do you think happens when we try to do addition with words?

```
> 'dog' + 'cat'  
dogcat  
> 'dog' + 5  
dog5  
> 34 + 5  
39  
> '34' + 5  
345  
> █
```

When we put something inside quotes, CoffeeScript treats it like a string of letters, even if it is all digits! That is why `'34' + 5` is `345`. Quoted values like this are called "strings."

The `Number()` function can be used to convert a string to a number, so that we can do ordinary arithmetic with it.

The `String()` function is opposite, and turns numbers into strings.

```
> Number('34') + 5  
39  
> String(34) + 5  
345  
> Number('dog') + 5  
NaN  
> █
```

If we try to convert a string to a number in a way that does not make sense, we get `NaN`, which stands for "Not a Number".


5. Creating Graphics

In Pencil Code, we can create graphics by using the turtle. There are five basic turtle functions:

code	meaning
pen red	chooses the pen color red
fd 100	moves forward by 100 pixels
rt 90	turns right by 90 degrees
lt 120	turns left by 120 degrees
bk 50	slides back by 50 pixels

In the test panel, enter two commands to draw a line:

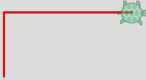
```
> pen red
> fd 50
> █
```



The reference at the end of this book lists many other colors that can be used. To stop drawing, use "pen null" to select no pen.

Try turning the turtle and drawing another line. Notice that `rt` turns the turtle in place, and we need to move the turtle with `fd` to draw a corner.

```
...
> rt 90
> fd 100
> █
```



Read about the `rt` function using `help`:

```
> help rt
rt(degrees) Right turn. Pivots clockwise by some degrees: rt 90
rt(degrees, radius) Right arc. Pivots with a turning radius:
    rt 90, 50
> █
```

If we give a second number to `rt`, the turtle will move while turning and form an arc. Try making a circle:

```
...
> rt 360, 30
> █
```

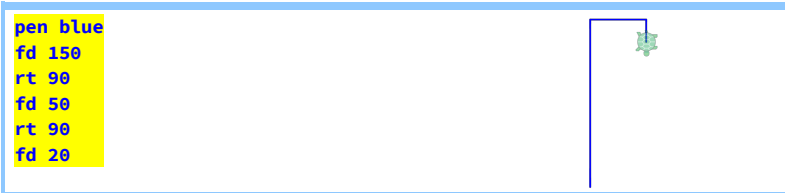


Remember to put a comma between the two numbers.

6. Making our First Program

We are ready to set up a hangman game. In the the editor on the left side of Pencil Code:

- Select and erase the example program text in the editor.
- Now type the following program into the editor.



```
pen blue
fd 150
rt 90
fd 50
rt 90
fd 20
```

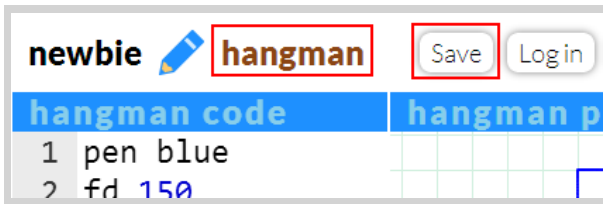
Press the triangular play button!

If it doesn't work, check the typing carefully and try again. Things to watch out for:

- Spell each function name correctly and in lowercase.
- Do not indent any of the lines of this program.
- Remember to put a space after the function names.

Each time we run the program, it clears the screen and starts again.

Now, rename the program from "first" to "hangman" by editing the name next to the pencil. Save it with the button at the top right.



A website will be created with your account name. If I choose the account name "newbie," a website is created at "newbie.pencilcode.net".


Once you have saved the program with the name "hangman," it is available at two different addresses on pencilcode:

- <http://yourname.pencilcode.net/edit/hangman> - this is where anyone can see and edit your program, but you need your password to save any changes.
- <http://yourname.pencilcode.net/home/hangman> - here is where you can share and run your program without showing the code.

7. Hurry Up and Wait

Write a welcome message after drawing the hangman shape:

```
pen blue
fd 150
rt 90
fd 50
rt 90
fd 20
write 'time to play hangman'
```




Notice that the Pencil Code Turtle is as slow as a turtle! Unless we speed it up with the *speed* function, the turtle takes its own slow time long after we have asked it to move, and the welcome message appears before the turtle is finished.

We can do two things to help with the slow turtle:

- Change the number of moves it makes per second using "*speed*."
- Ask the program to wait for the turtle, using "*await done defer*()."

```
speed 10
pen blue
fd 150
rt 90
fd 50
rt 90
fd 20
await done defer()
write 'time to play hangman'
```



Now the turtle moves faster, and the program waits until the turtle is done before writing the welcome message.

A couple things to know:

- Do not use a space between *defer* and the parentheses "*defer*()".
- We can make the turtle move instantly by using "*speed* Infinity".

Even if you have programmed before, *await/defer* may be new to you. These keywords create *continuations*, and they are part of Iced CoffeeScript. To explore how they work in more detail, look up Max Krohn's Iced CoffeeScript page online.

8. Using "for" to Repeat

We can repeat steps in a program with the "for" command.

Try adding three lines to the end of our program so that it looks like this:

```
write 'time to play hangman'  
secret = 'crocodile'  
for letter in secret  
  write letter
```

You should see this:

```
time to play hangman  
c  
r  
o  
c  
o  
d  
i  
l  
e
```

The program is saying: for every letter in the secret, write letter. So the computer repeats "write letter" nine times, once for each letter.

If it doesn't work, check the program and make sure the line after the **for** is indented; that is how CoffeeScript knows which line to repeat.

Once you have the hang of it, keep the word secret by changing the program to write underscores instead of letters:

```
write 'time to play hangman'  
for letter in secret  
  append '_'
```

Notice how "**append**" instead of "**write**" puts text on the same line instead of starting a new line each time:

```
time to play hangman  
-----
```

9. Using "if" to Choose

In our hangman game, we should show where any guessed letters are. To decide whether to print a blank line or a letter, we will need to use "if" and "else".

Add four new lines to our program:

```
write 'time to play hangman'  
secret = 'crocodile'  
hints = 'aeiou'  
  
for letter in secret  
    if letter in hints  
        append letter + ' '  
    else  
        append '_ '
```

Don't forget to line everything up, and remember to save it.

What happens when you run it? It reveals all the letters in "hints": all the vowels.

Our screen looks like this:

```
time to play hangman  
_ _ o _ o _ i _ e
```

Here is how it works.

The line "**if** letter in hints" makes a choice.

- If the letter is among our hints, it appends the letter together with a space after it.
- Otherwise ("else") it appends a little underscore with a space after it.

Since the whole thing is indented under the "**for** letter in secret," this choice is repeated for every letter.

Check the spelling and spacing and punctuation if you get errors. Take your time to get it to work.

10. Input with "read"

Our game is no good if players can't guess. To let the player guess we will use a function called "read"

It works like this:

```
await read defer guess
```

This shows an input box and puts the program on hold until the user enters a value for "guess".

The "await" and "defer" commands work together to pause and resume the program while waiting for an answer to be entered.

- **await** tells the program to pause after starting the **read** function.
- **defer** tells **read** what to do after it is done: it continues the program after saving the answer as "guess."

Try adding two lines to the program to add an **await read**, like this:

```
write 'time to play hangman'  
secret = 'crocodile'  
hints = 'aeiou'  
write 'guess a letter'  
await read defer guess  
hints += guess  
for letter in secret  
  if letter in hints  
    append letter + ' '  
  else  
    append '_ '
```

The "hints += guess" line adds the guess to the string of hints. If the string of hints was "aeiou" and the new guess is "c", then the string of hints will become "aeiouc".

Let's run it.

```
time to play hangman  
guess a letter  
⇒ c  
c_o_o_i_e
```

When we run the program, it will show us where our guessed letter appears.

11. Using "while" to Repeat

We need to let the player take more than one turn.

The "while" command can repeat our program until the player is out of turns.

```
write 'time to play hangman'
secret = 'crocodile'
hints = 'aeiou'
turns = 5

while turns > 0
  for letter in secret
    if letter in hints
      append letter + ' '
    else
      append '_'

write 'guess a letter'
await read defer guess
hints += guess
turns -= 1
```

Indent everything under the "while" command to make this work. The editor will indent a whole block of code if you select it all at once and press the "Tab" key on the keyboard. "Shift-Tab" will unindent code.

Also move the guessing after the hint instead of before.

The command "turns -= 1" means subtract one from "turns," so if it used to be 5, it will be 4. Then the next time around it will be 3 and so on. When turns is finally zero, the "while" command will stop repeating.

Try running the program. Does it work?

Any time we want to see the value of a variable, we can type its name into the test panel.

```
test panel (type help for help)
> hints
aeioucsn
> turns
2
> █
```

12. Improving our Game

We can already play our game. Now we should fix it up to make it fun.

- The player should win right away when there are no missing letters.
- The player should only lose a turn on a wrong guess.
- When the player loses, the game should tell the secret.

Here is one way to improve it.

```
write 'time to play hangman'
secret = 'crocodile'
hints = 'aeiou'
turns = 5

while turns > 0
  blanks = 0
  for letter in secret
    if letter in hints
      append letter + ' '
    else
      append '_'
      blanks += 1

  if blanks is 0
    write 'You win!'
    break

  write 'guess a letter'
  await read defer guess
  hints += guess

  if guess not in secret
    turns -= 1
    write 'Nope.'
    write turns + ' more turns'
    if turns is 0
      write 'The answer is ' + secret
```

Each time the word is printed, the "blanks" number starts at zero and counts up the number of blanks. If it ends up at zero, it means there are no blanks. So the player has guessed every letter and has won! In that case, no more guesses are needed, so the "break" command breaks out of the "while" section early.

The "if guess not in secret" line checks if the guess was wrong. We only count down the "turns" if our guess was wrong.

When we guess wrong, we also print a bunch of messages like "Nope" and how many more turns we have. When we are wrong for the last time we print the secret.

13. Making it Look Like Hangman

It will be more fun if we make our game look like Hangman.

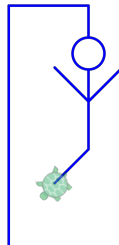
All we need to do is draw parts of the poor hangman person when there is a wrong guess. Try adding something like this to the wrong guess part:

```
...
write 'Nope.'
write turns + ' more turns'
if turns is 4 then lt 90; rt 540, 10; lt 90
if turns is 3 then fd 20; lt 45; bk 30; fd 30
if turns is 2 then rt 90; bk 30; fd 30; lt 45; fd 30
if turns is 1 then rt 45; fd 30
if turns is 1 then fd 30
if turns is 0
bk 30; lt 90; fd 30
await done defer()
write 'The answer is ' + secret
```

The semicolons (;) are just a way to put more than one step on the same line. Notice when putting the "if" on the same line as the commands to run, we must use the word "then" between the test and the commands.

Try making variations on the hangman drawings for each step.

Whenever we want to pause the program to wait for the turtle to finish drawing, we can use "await done defer()". This pauses the program and tells the **done** function to resume the program after drawing has completed.



14. Picking a Random Secret

The only problem with the game is that it always plays the same secret word. We should use the *random* function to choose a random word.

Change the line that sets the secret so that it looks like this:

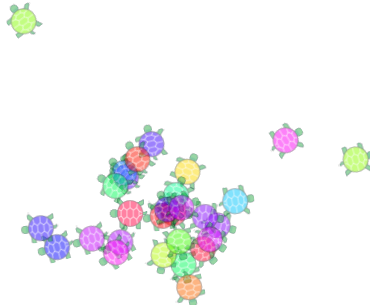
```
...
write 'time to play hangman'
secret = random ['tiger', 'panda', 'mouse']
hints = 'aeiou'
...
```

The square brackets [] and commas make a list, and the random function picks one thing randomly from the list.

Of course, we can make the list as long as we like. Here is a longer list:

```
...
write 'time to play hangman'
secret = random [
'crocodile'
'elephant'
'penguin'
'pelican'
'leopard'
'hamster'
]
...
```

We can write a long list on lots of lines like this, as long as we remember to end any brackets [] that we started. When we list items on their own lines, the commas are optional.



15. Loading a List from the Internet

There is a longer list of animals on the internet at the address <http://pencilcode.net/data/animals>.

We can load this data in CoffeeScript using a jQuery function "\$.get". (The \$ is the [jQuery library](#), and it has more than one hundred functions that are useful for web apps. Read more about jQuery at learn.jquery.com.)

The code looks like this:

```
...
write 'time to play hangman'
await $.get 'http://pencilcode.net/data/animals', defer animals
secret = random animals.split '\n'
...
```

What this means is:

```
await $.get 'http://pencilcode.net/data/animals', defer animals
```

Pause the program until the \$.get is done.

```
await $.get 'http://pencilcode.net/data/animals', defer animals
```

Open up the address <http://pencilcode.net/data/animals>

```
await $.get 'http://pencilcode.net/data/animals', defer animals
```

Tell \$.get to resume the program after putting the answer in "animals."

```
secret = random animals.split '\n'
```

The special string "\n" is the newline character between lines in a file.

```
secret = random animals.split '\n'
```

Split the animals string into an array, with one entry per line.

```
secret = random animals.split '\n'
```

Choose one item from the array randomly.

```
secret = random animals.split '\n'
```

Call this random word "secret".

16. The Whole Hangman Program

Here is the whole program from beginning to end:

```
speed 10
pen blue
fd 150
rt 90
fd 50
rt 90
fd 20
await done defer()
write 'time to play hangman'
await $.get 'http://pencilcode.net/data/animals', defer animals
secret = random animals.split '\n'
hints = 'aeiou'
turns = 5

while turns > 0
  blanks = 0
  for letter in secret
    if letter in hints
      append letter + ' '
    else
      append '_'
      blanks += 1

  if blanks is 0
    write 'You win!'
    break

  write 'guess a letter'
  await read defer guess
  hints += guess

  if guess not in secret
    turns -= 1
    write 'Nope.'
    write turns + ' more turns'
    if turns is 4 then lt 90; rt 540, 10; lt 90
    if turns is 3 then fd 20; lt 45; bk 30; fd 30
    if turns is 2 then rt 90; bk 30; fd 30; lt 45; fd 30
    if turns is 1 then rt 45; fd 30
    if turns is 0
      bk 30; lt 90; fd 30
      await done defer()
      write 'The answer is ' + secret
```

17. Making it Yours

The best part of programming is adding your own personal style.

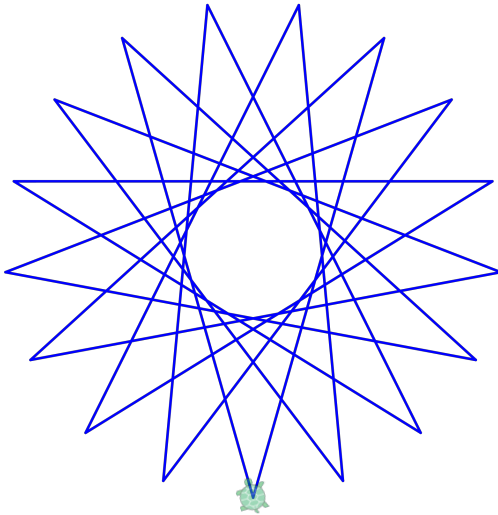
Try making the game so that it plays again automatically after you are done. Can you make the game harder or easier? Can you give the player a reward for winning?

Be sure to explore the functions in the online help, and experiment with the examples in the remainder of this book. They will be a source of ideas.

For example, take a look at using sound effects and music. Try exploring the "play" function, and search the internet to learn about ABC notation, chords, waveforms, and ADSR envelopes.

Sometimes the simplest ideas can make a big difference. The "ct()" function clears the text on the screen and the "cg()" function clears the graphics. Maybe this could be used to make a two-player game where one person comes up with the secret word, or where two players compete to guess the word first.

You will quickly find that the real challenge of programming is not in the code. The real challenge is in putting your imagination into the code.



Reference

Movement

fd 50 forward 50 pixels
 bk 10 backward 10 pixels
 rt 90 turn right 90 degrees
 lt 120 turn left 120 degrees
 home() go to the page center
 slide x, y slideright x and forward y
 moveto x, y go to x,y relative to home
 turnto 45 set direction to 45 (NE)
 turnto obj point toward obj
 speed 30 do 30 moves per second

Appearance

ht() hide the turtle
 st() show the turtle
 scale 8 do everything 8x bigger
 wear yellow wear a yellow shell
 fadeOut() fade and hide the turtle
 remove() totally remove the turtle

Output

write 'hi' adds HTML to the page
 p = write 'fast' remembers written HTML
 p.html 'quick' changes old text
 button 'go', adds a button with
 -> fd 10 an action
 read (n) -> adds a text input with
 write n*n an action
 t = table 3,5 adds a 3x5 <table>
 t.cell(0, 0). selects the first cell of the
 text 'aloha' table and sets its text

Other Objects

\$(window) the visible window
 \$('p').eq(0) the first <p> element
 \$('#zed') the element with id="zed"

Drawing

pen blue draw in blue
 pen red, 9 9 pixel wide red pen
 pen null use no color
 pen off pause use of the pen
 pen on use the pen again
 mark 'X' mark with an X
 dot green draw a green dot
 dot gold, 30 30 pixel gold circle
 pen 'path' trace an invisible path
 fill cyan fill traced path in cyan

Properties

turtle name of the main turtle
 getxy() [x, y] position relative to home
 direction() direction of turtle
 hidden() if the turtle is hidden
 touches(obj) if the turtle touches obj
 inside(window) if enclosed in the window
 lastmousemove where the mouse last moved

Sets

g = hatch 20 hatch 20 new turtles
 g = \$('img') select all as a set
 g.p[an (j) -> direct the jth turtle to go
 @fd j * 10 forward by 10j pixels

Other Functions

see obj inspect the value of obj
 speed 8 set default speed
 rt 90, 50 90 degree right arc of radius 50
 tick 5, -> fd 10 go 5 times per second
 click -> fd 10 go when clicked
 random [3,5,7] return 3,5, or 7
 random 100 random[0.99]
 play 'ceg' play musical notes

Colors

white	gainsboro	silver	darkgray	gray	dimgray	black
whitesmoke	lightgray	lightcoral	rosybrown	indianred	red	maroon
snow	mistyrose	salmon	orangered	chocolate	brown	darkred
seashell	peachpuff	tomato	darkorange	peru	firebrick	olive
linen	bisque	darksalmon	orange	goldenrod	sienna	darkolivegreen
oldlace	antiquewhite	coral	gold	limegreen	saddlebrown	darkgreen
floralwhite	navajowhite	lightsalmon	darkkhaki	lime	darkgoldenrod	green
cornsilk	blanchedalmond	sandybrown	yellow	mediumseagreen	olivedrab	forestgreen
ivory	papayawhip	burlywood	yellowgreen	springgreen	seagreen	darkslategray
beige	moccasin	tan	chartreuse	mediumspringgreen	lightsagegreen	teal
lightyellow	wheat	khaki	lawngreen	aqua	darkturquoise	darkcyan
lightgoldenrodyellow	lemonchiffon	greenyellow	darkseagreen	cyan	deepskyblue	midnightblue
honeydew	palegoldenrod	lightgreen	mediumaquamarine	cadetblue	steelblue	navy
mintcream	palegreen	skyblue	turquoise	dodgerblue	blue	darkblue
azure	aquamarine	lightskyblue	mediumturquoise	lightslategray	blueviolet	mediumblue
lightcyan	paleturquoise	lightsteelblue	cornflowerblue	slategray	darkorchid	darkslateblue
aliceblue	powderblue	thistle	mediumslateblue	royalblue	fuchsia	indigo
ghostwhite	lightblue	plum	mediumpurple	slateblue	magenta	darkviolet
lavender	pink	violet	orchid	mediumorchid	mediumvioletred	purple
lavenderblush	lightpink	hotpink	palevioletred	deeppink	crimson	darkmagenta