Lepidus: What manner o' thing is your crocodile?
Antony: It is shap'd, sir, like itself, and it is as broad as it hath breadth; it is just as high as it is, and moves with its own organs. It lives by that which nourisheth it, and the elements once out of it, it transmigrates.
Lepidus: What color is it of?
Antony: Of its own color too.
Lepidus: 'Tis a strange serpent.
Antony: 'Tis so. And the tears of it are wet.
What is a quine?

A computer program that accepts no input (including, critically, from the filesystem), and produces as its only output a copy of its own source code.

Ken Thompson once said that figuring out this puzzle "is a revelation that far surpasses any benefit obtained by being told how to do it."

If you want to try to work it out on your own before we spoil the ending, walk out now.
What's the problem? Printing isn't hard...

Well, because it's so obviously impossible.

Let me explain by way of dialogue, stolen shamelessly from WikiWikiWeb (the very first Wiki).
Can you print "Hello, world."?

Easily.

print("Hello, world.")

So now you know how to print *anything*.

*A few things are a bit tricky, but yes -- anything.*
Ah, but can you print quotes around it?

*It's a bit tricky, but not a problem.*

```python
print("\"Hello, world.\"\")
```

You've just replaced the problem of the quotes with the problem of the slash.
Quines are impossible.

No, I can print slashes as well:

```python
print("quote: \\" slash: \\ another slash: \\
another quote: \\" The end.")
```

Ah, but I bet you can't print the entire program.

Sure I can.
Quines are impossible.

```python
print("print(\"Hello, world.\")")
```

Yes, that prints the hello world program. But ...

```
print("print(\"print(\"Hello, world.\"\")\")")
```

Oh, I see what you want.

```
print("print(\"print(\"Hello, world.\"\")\")")
```
Yes, that prints *a* program. But it doesn't print *itself*.

*It's a bit awkward, but I suppose you want this:*

```python
print("print("print("print("Hello, world."))")")
```
Yes, that prints *a* program. But it doesn't print *itself*.

... Yes, that prints *a* program. But it doesn't print *itself*.

Oh. But that would ... that would require an infinite number of escapes.
print(" <some stuff> <infinite number of slashes >"Hello, world. <infinite number of slashes>" <more stuff> ")

Sorry, my computer doesn't have an infinite amount of memory.

Well then. It's impossible for a finite-size source file to hold another copy of the source file inside itself. A thing cannot be bigger than itself.

Obviously.
So... These programs are impossible?

Of course not. We wouldn't be having a tutorial on them if they were impossible.

You just have to be clever.
So what does a working quine look like?

We turn to the 1994 IOCCC (International Obfuscated C Coding Contest) Winner for "Worst Abuse of the Rules", shown on the right.
It's an empty file. When you run and compile and empty file (at least in some of the C compilers of the day...), you get a program that does nothing.

So it's technically a quine, which is the best kind of quine.

... But that's not a very satisfying answer, is it?
Haha. So, seriously. How do I do the thing?

Here you go, a very nice Quine in the C programming language by Aldo Cortesi:

```c
int main(void){
    char str[]="int main(void){ char str[]= %c%s%c; printf(str, 0x22, str, 0x22);}"
    printf(str, 0x22, str, 0x22);
}```
What's going on here?

In my way of thinking, there are two problems that make Quines hard:

1. The actual printing the source code, which you can do by basically printing it "twice" - the first time inside a string, and the second time outside it.
2. Escapes, which make the code different inside and outside.

Here we use `printf` to make a template that we can then use to print the code "twice". Also, we use 0x22, the ASCII code for the double-quote character, to escape the quoting trap.
OK, cool. But what does it mean?

Quines seem like a minor exercise in syntax, but their existence is actually a consequence of some rather serious CS theorems.

Without getting into the technical details (which are a part of the course on Computability), we'll say that quines are proven to exist as a consequence of Kleene's recursion theorem that lets us assume that every given turing machine M has access to the encoding of itself.

A quine is an example of a "fixed point", or an such that F(x)=x. The function F here is the compiler & execution environment of the language.
Are quines useful?

Not really. But they're very cool!

Quines were used in Ken Thompson's seminal "Reflections on Trusting Trust" article as a way of hiding malware in the C compiler.

Apparently quines are also used in theoretical research on self repairing code.