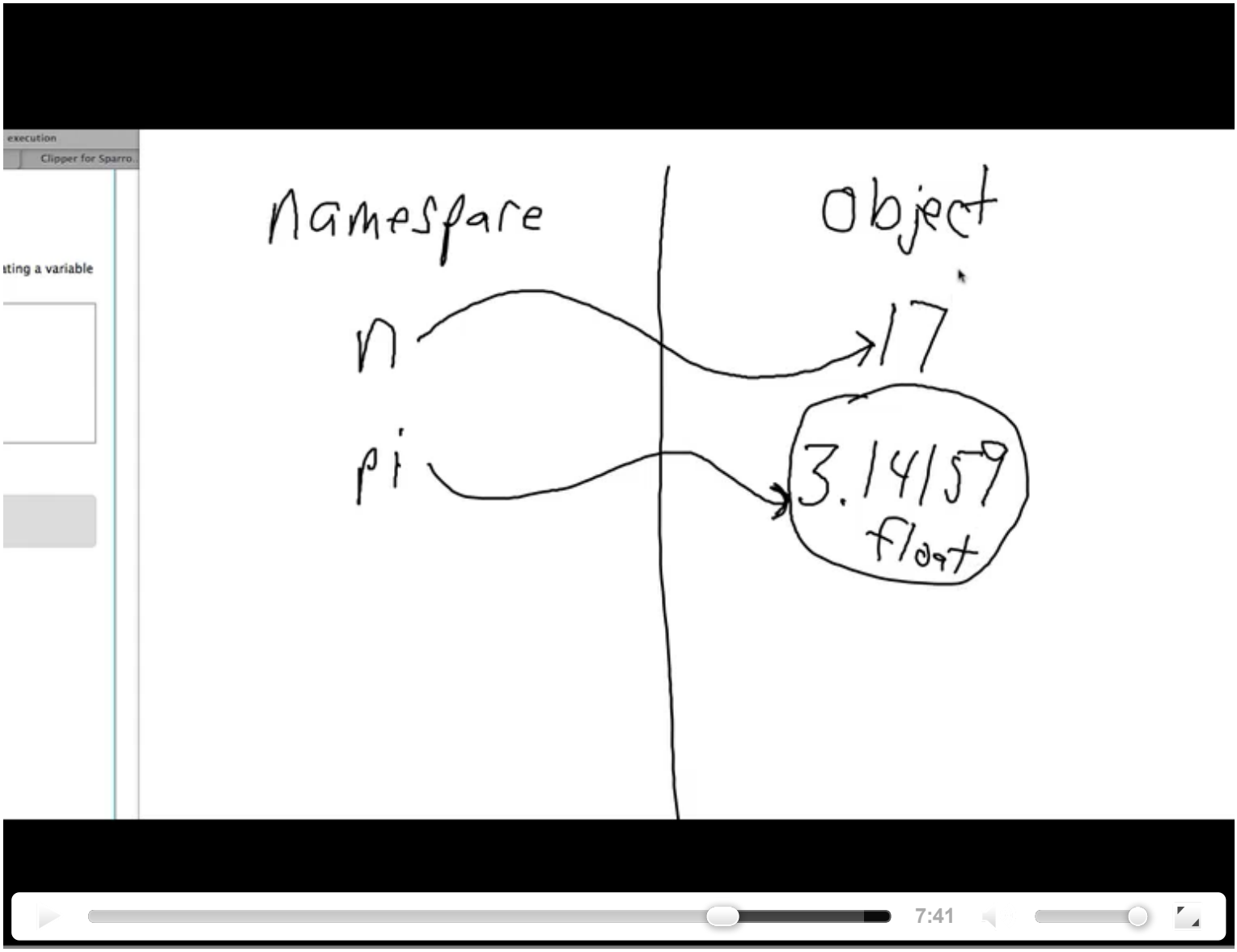
Variables



One of the most powerful features of a programming language is the ability to manipulate **variables**. A variable is a name that refers to a value.

Assignment statements create new variables and also give them values to refer to.

```
message = "What's up, Doc?"
n = 17
pi = 3.14159
```

This example makes three assignments. The first assigns the string value `"What's up, Doc?"` to a new variable named `message`. The second gives the integer `17` to `n`, and the third assigns the floating-point number `3.14159` to a variable called `pi`.

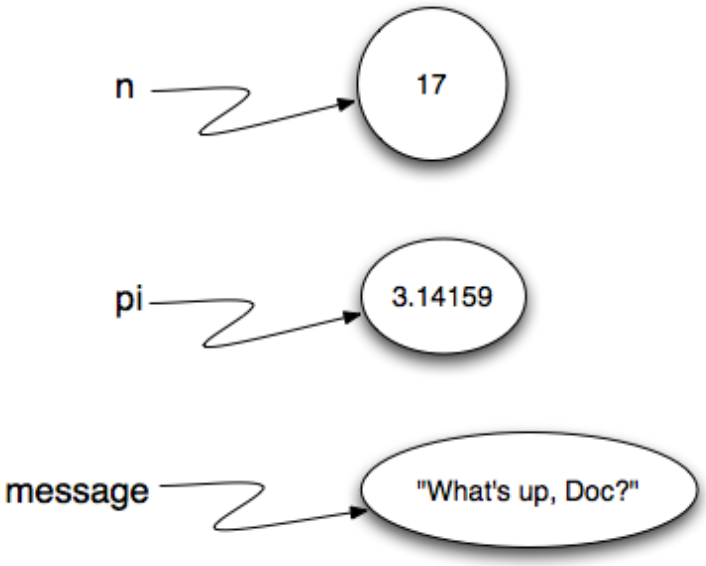
The **assignment token**, `=`, should not be confused with *equality* (we will see later that equality uses the `==` token). The assignment statement links a *name*, on the left hand side of the operator, with a *value*, on the right hand side. This is why you will get an error if you enter:

```
17 = n
```

Tip

When reading or writing code, say to yourself “n is assigned 17” or “n gets the value 17” or “n is a reference to the object 17” or “n refers to the object 17”. Don’t say “n equals 17”.

A common way to represent variables on paper is to write the name with an arrow pointing to the variable’s value. This kind of figure, known as a **reference diagram**, is often called a **state snapshot** because it shows what state each of the variables is in at a particular instant in time. (Think of it as the variable’s state of mind). This diagram shows the result of executing the assignment statements shown above.



If you ask Python to evaluate a variable, it will produce the value that is currently linked to the variable. In other words, evaluating a variable will give you the value that is referred to by the variable.

```
1 message = "What's up, Doc?"
2 n = 17
3 pi = 3.14159
4
5 print(message)
6 print(n)
7 print(pi)
8
```

ActiveCode: 1 (ch02_9)

Run

```
What's up, Doc?
17
3.14159
```

In each case the result is the value of the variable. To see this in even more detail, we can run the program using codelens.

```
1 message = "What's up, Doc?"
2 n = 17
3 pi = 3.14159
4
5 print(message)
6 print(n)
7 print(pi)
```

<< First | < Back | Program terminated | Forward > | Last >>

line that has just executed

next line to execute

Program output:

```
What's up, Doc?
17
3.14159
```

Frames Objects

Global variables

message	"What's up, Doc?"

n	17
pi	3.1416

CodeLens: 1 (ch02_9_codelens)

Now, as you step thru the statements, you can see the variables and the values they reference as those references are created.

Variables also have types; again, we can ask the interpreter what they are.

```
message = "What's up, Doc?"
n = 17
pi = 3.14159

print(type(message))
print(type(n))
print(type(pi))

```

ActiveCode: 2 (ch02_10)

Run

```
<type 'str'>
<type 'int'>
<type 'float'>
```

The type of a variable is the type of the object it currently refers to.

We use variables in a program to “remember” things, like the current score at the football game. But variables are *variable*. This means they can change over time, just like the scoreboard at a football game. You can assign a value to a variable, and later assign a different value to the same variable.

Note

This is different from math. In math, if you give x the value 3, it cannot change to refer to a different value half-way through your calculations!

To see this, read and then run the following program. You’ll notice we change the value of day three times, and on the third assignment we even give it a value that is of a different type.

```
1  day = "Thursday"
2  print(day)
3  day = "Friday"
4  print(day)
5  day = 21
6  print(day)
```

<< First | < Back | Program terminated | Forward > | Last >>

line that has just executed

next line to execute

Program output:

```
Thursday
Friday
21
```

Frames Objects

Global variables

day	21

CodeLens: 2 (ch02_11)

A great deal of programming is about having the computer remember things. For example, we might want to keep track of the number of missed calls on your phone. Each time another call is missed, we will arrange to update or change the variable so that it will always reflect the correct value.

Check your understanding

sdat-4: What is printed when the following statements execute?

```
day = "Thursday"
day = 32.5
day = 19
print(day)
```

- a) Nothing is printed. A runtime error occurs.
- b) Thursday
- c) 32.5
- d) 19

Check Me | Compare Me

Correct!! The variable day will contain the last value assigned to it when it is printed.

Variable Names and Keywords

Variable names can be arbitrarily long. They can contain both letters and digits, but they have to begin with a letter or an underscore. Although it is legal to use uppercase letters, by convention we don't. If you do, remember that case matters. `Bruce` and `bruce` are different variables.

Caution

Variable names can never contain spaces.

The underscore character (`_`) can also appear in a name. It is often used in names with multiple words, such as `my_name` or `price_of_tea_in_china` . There are some situations in which names beginning with an underscore have special meaning, so a safe rule for beginners is to start all names with a letter.

If you give a variable an illegal name, you get a syntax error. In the example below, each of the variable names is illegal.

```
76trombones = "big parade"
more$ = 1000000
class = "Computer Science 101"
```

`76trombones` is illegal because it does not begin with a letter. `more$` is illegal because it contains an illegal character, the dollar sign. But what's wrong with `class` ?

It turns out that `class` is one of the Python **keywords**. Keywords define the language's syntax rules and structure, and they cannot be used as variable names. Python has thirty-something keywords (and every now and again improvements to Python introduce or eliminate one or two):

and	as	assert	break	class	continue
def	del	elif	else	except	exec
finally	for	from	global	if	import
in	is	lambda	nonlocal	not	or
pass	raise	return	try	while	with
yield	True	False	None		

You might want to keep this list handy. If the interpreter complains about one of your variable names and you don't know why, see if it is on this list.

Programmers generally choose names for their variables that are meaningful to the human readers of the program — they help the programmer document, or remember, what the variable is used for.

Caution

Beginners sometimes confuse "meaningful to the human readers" with "meaningful to the computer". So they'll wrongly think that because they've called some variable `average` or `pi`, it will somehow automagically calculate an average, or automagically associate the variable `pi` with the value 3.14159. No! The computer doesn't attach semantic meaning to your variable names.

So you'll find some instructors who deliberately don't choose meaningful names when they teach beginners — not because they don't think it is a good habit, but because they're trying to reinforce the message that you, the programmer, have to write some program code to calculate the average, or you must write an assignment statement to give a variable the value you want it to have.

Check your understanding

sdat-5: True or False: the following is a legal variable name in Python: A_good_grade_is_A+

a) True

b) False

Check Me | Compare Me

Correct!! - The + character is not allowed in variable names (everything else in this name is fine).