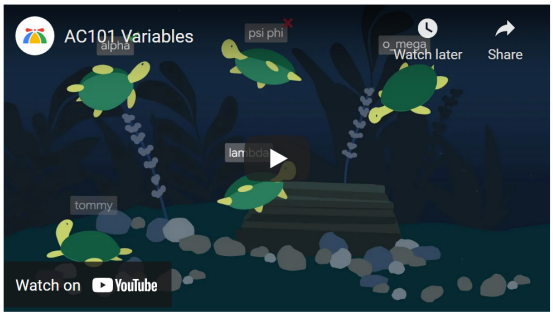
2.7. Variables



Activity: 1 -- Video: (assignvid)

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 assigns the integer `17` to `n`, and the third assigns the floating-point number `3.14159` to a variable called `pi`.

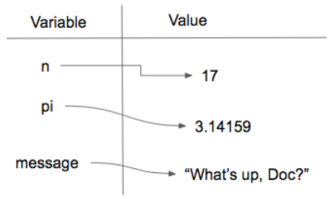
The **assignment token**, `=`, should not be confused with *equality* (we will see later that equality uses the `==` token). The assignment statement links a *name*, on the left hand side of the operator, with a *value*, on the right hand side. This is why you will get an error if you enter:

```
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 your program includes a variable in any expression, whenever that expression is executed it will produce the value that is linked to the variable at the time of execution. In other words, evaluating a variable looks up its value.

Save & Run 4/17/2021, 12:34:21 PM - 2 of 2 Show in CodeLens

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

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

Activity: 2 -- ActiveCode (ac2_7_1)

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)
```

Global frame

message	"What's up, Doc?"
n	17
pi	3.1416

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

line that has just executed
next line to execute

Program output:

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

Activity: 3 -- CodeLens: (clens2_7_1)

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

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 algebra, 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.

Python 3.3

Frames Objects

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

Global frame

day	21

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

line that has just executed
next line to execute

Program output:

```
Thursday
Friday
21
```

Activity: 4 -- CodeLens: (clens2_7_2)

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.

Any place in a Python program where a number or string is expected, you can put a variable name instead. The python interpreter will substitute the value for the variable name.

For example, we can find out the data type of the current value of a variable by putting the variable name inside the parentheses following the function name `type`.

Save & Run | 4/17/2021, 12:35:01 PM - 2 of 2 | Show in CodeLens

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

```
<class 'str'>
<class 'int'>
<class 'float'>
```

Activity: 5 -- ActiveCode (ac2_7_2)

Note

If you have programmed in another language such as Java or C++, you may be used to the idea that *variables* have types that are declared when the variable name is first introduced in a program. Python doesn't do that. Variables don't have types in Python; *values* do. That means that it is acceptable in Python to have a variable name refer to an integer and later have the same variable name refer to a string. This is almost never a good idea, because it will confuse human readers (including you), but the Python interpreter will not complain.

Check your understanding

data-7-1: 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

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

Activity: 6 -- Multiple Choice (question2_7_1)

You have attempted 6 of 6 activities on this page

2.6. Type conversion functions">
e conversion functions">
2.8. Variable Names and Keywords">

✔ Completed. Well Done!

| Back to top