Values, Variables and Assignment

Goal

The goal of this lesson is to understand the difference between values and variables, and how to use them in assignments.

Objectives

- The student will be able to write a literal value for the three primitive types: number, boolean and string.
- The student will be able to distinguish between integer values and floating point values.
- The student will be able to compose a valid assignment statement with variable names and literal values.

Lesson

These concepts will be given interactively, via whiteboard/projector. Students can take notes during the discussion. Estimated time ~15 minutes.

The grammar of imperative languages is composed of statements, expressions, and values. Today we will discuss values.

A *value* is a specific *thing* with a specific *type*.

Simple (primitive) value types are either numbers, strings or booleans.

There are two kinds of numbers, integers and floating point numbers.

Floating point numbers have a period to indicate their fractional part, or an 'e' to indicate it's in exponential notation. Floating point numbers are *not* real numbers.

Neither integers nor floating point numbers ever contain commas.

Values are either *literal* or *variable*.

We can use the `print` statement to print a literal or variable. We can use the `type` statement to get the type of a value.

A literal value never changes, it is explicitly represented.

```
3               # an integer literal value
print(type(3))  # prints 'int'

4.002               # a floating point literal value
print(type(4.002))  # prints 'float'
12e3        # another floating point literal value

False       # a boolean literal value
```

```
"fred"      # a string literal value
'fred'      # another string literal value
```

Notice that string literals are surrounded by either single ' or double " quotes.

Boolean values can only be true or false. The literal values for these are *True* and *False*.

A variable is a name for a value that can change. The variable can only have one specific value at time, but can be reassigned different values. We use an *assignment* statement to both introduce a new variable name and to reassign an existing name.

An assignment statement has a variable name on the left followed by a single = symbol, then a valid value.

```
a = 5
a = 7
print(a)        # prints 7
print(type(a))  # prints 'int'
```

A variable name must be a valid *identifier*. Identifiers are words that mean something in Python. Variables are one kind of identifier, another are the reserved words that make up the language (there are 33).

Here are the reserved words:

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

Identifiers must be at least a single character long, must contain only letters (upper or lowercase), digits or the underscore character (_) and must not begin with a digit. Variable must be valid identifiers, and my not be one of the reserved words.

It is convention to use the underscore to separate words (snake-case). Here are some variable assignments.

```
current_temp = 45.6
age = 12
my_brothers_name = 'John'
```

One way we can set a variable's value is by taking user input. This is done by using the input function.

```
age = input('Enter a number for age')
print("You typed: ", age)
```

Note the value that comes back from the input function will always be a string, possibly empty.

To turn a number into a string, we can use `int` or `float` as a function.

```
age = input("Enter your age")
older = age + int(age)
```

Activities

Students can work through these activities directly on their Pi. Estimated time ~ 20 minutes.

1. Type in the following program, what does it do? Add a variable `y`, give it a value and have the print statement print the value as well (on the same line).

```
x = 5
print('x is', x)
```

2. Which of these statements are valid? Type them in to verify, and pay attention to the errors that occur. Try to fix the invalid statements so they do not cause an error.

```
5 = "5"
"5" = 5
five = "5"
3_three = 3
three = "three"
```

3. What do you think will be printed from running this? Type it in to verify.

```
a = 6
b = a
print('c is now', c)
c = b + 1
c = c - a
print('c is now', c)
c = 0
print('c is now', c)
```

4. We can use the `round` statement round a floating point number to the nearest integer value. What do you think this program will do? Type it in to verify.

```
a = 6.5
b = 11e-1
```

```
print(round(a + b))
```

5. Rather than round, we can use the Math functions floor or ceil to change a floating point number to an integer. Since these functions are part of the Math module we have to import it first. Try this:

```
import Math
a = 4.7
print(Math.floor(a))
print(Math.ceil(a))
```

Verify that round, floor and ceil all return integers, how would you do that?

6. Notice that our `print` statement is not a reserved word. That means we can create a variable named `print`. But is this a good idea? Try it.

```
print = 7
print(print)
```

What happens if you try this with a reserved word, like `if` ?

7. Write a program that will convert a number entered by the user from a temperature in fahrenheit to celsius. Only print the answer with a single decimal of precision.

Hint: one way to get a float to single precision is to pass another argument to round, `round(a, 1)`. There are others, feel free to use them.