Week 2

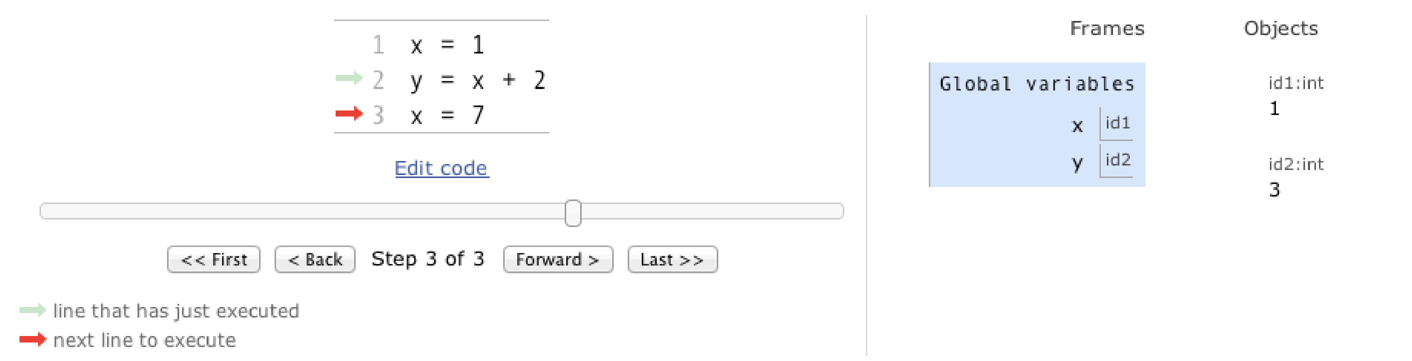
Video 1: Visualizing Assignment Statements

On the Resources page is a link to a Python Visualizer that follows the model we use to draw pictures of computer memory.

Consider this code:

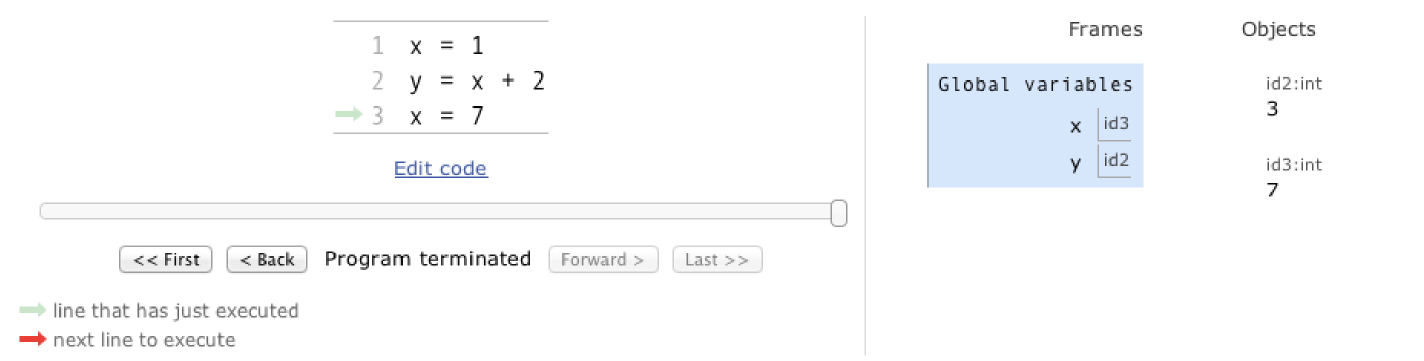
```
x = 1
y = x + 2
x = 7
```

When we trace this in the visualizer and click button Forward twice, this is the result:



```
x = 1
y = x + 2
x = 7
```

Clicking Forward once more results in this:



```
x = 1
y = x + 2
x = 7
```

Notice that y's value did not change during this step.

Here is a link to the Python Visualizer containing this code so that you can explore this yourself. **We strongly encourage you to step forward and backward through this program until you understand every step of execution.**

Exercise shown in Python Visualizer (Start yourself)

Video 2: Type str: Strings in Python

String Literal

A *string literal* is a sequence of characters. In Python, this type is called `str`. Strings in Python start and end with a single quotes (') or double quotes ("). A string can be made up of letters, numbers, and special characters. For example:

```
>>> 'hello'
'hello'
>>> 'how are you?'
'how are you?'
>>> 'short- and long-term'
```

```
short- and long-term
```

If a string begins with a single quote, it must end with a single quote. The same applies to double-quoted strings. You can not mix the type of quotes.

Escape Sequences

To include a quote within a string, use an *escape character* (\) before it. Otherwise Python interprets that quote as the end of a string and an error occurs. For example, the following code results in an error because Python does not expect anything to come after the second quote:

```
>>> storm_greeting = 'wow, you're dripping wet.'
SyntaxError: invalid syntax
```

The *escape sequence* \' indicates that the second quote is simply a quote, not the end of the string:

```
>>> storm_greeting = 'Wow, you\'re dripping wet.'
"Wow, you're dripping wet."
```

An alternative approach is to use a double-quoted string when including a a single-quote within it, or vice-versa. Single- and double-quoted strings are equivalent. For example, when we used double-quotes to indicate the beginning and end of the string, the single-quote in `you're` no longer causes an error:

```
>>> storm_greeting = "Wow, you're dripping wet."
"Wow, you're dripping wet."
```

String Operators

Expression	Description	Example	Output
`str1 + str2`	concatenate `str1` and `str1`	`print('ab' + 'c')`	`abc`
`str1 * int1`	concatenate `int1` copies of `str1`	`print('a' * 5)`	`aaaaa`
`int1 * str1`	concatenate `int1` copies of `str1`	`print(4 * 'bc')`	`bcbcbcbc`

Note: *concatenate* means to join together

The `*` and `+` operands obey by the standard precedence rules when used with strings.

All other mathematical operators and operands result in a `TypeError`.

Optional Book Reading

Chapter 4.1 Creating Strings of Characters
Chapter 4.2 Using Special Characters in Strings

Video 3: Input/Output and `str` Formatting

Function `print`

Python has a built-in function named `print` that displays messages to the user. For example, the following function call displays the string `"hello"`:

```
>>> print("hello")
hello
```

In the output above, notice that `hello` is displayed without the quotation marks. The quotes are only for Python's internal string formatting and are not seen by the user.

The `print` function may also be called with a mathematical expression for an argument. Python evaluates the mathematical expression first and then displays the resulting value to the user. For example:

```
>>> print(3 + 7 - 3)
7
```

Finally, `print` can take in more than one argument. Each pair of arguments is separated by a comma and a space is inserted between them when they are displayed. For example:

```
>>> print("hello", "there")
hello there
return vs. print
```

Recall: The general form of a `return` statement:
`return expression`

When a `return` statement executes, the expression is evaluated to produce a memory address.

- What is passed back to the caller?
  That memory address is passed back to the caller.
- What is displayed?
  Nothing!

An example of `return`:

```
>>> def square_return(num):
    return num ** 2
>>> answer_return = square_return(4)
>>> answer_return
16
```

The general form of a `print` function call:

```
print(arguments)
```

When a `print` function call is executed, the argument(s) are evaluated to produce memory address(es).

- What is passed back to the caller?
  Nothing!
- What is displayed?
  The values at those memory address(es) are displayed on the screen.

An example of `print`:

```
>>> def square_print(num):
    print("The square of num is", num ** 2)
>>> answer_print = square_print(4)
The square num is 16
>>> answer_print
>>>
```

Function `input`

The function `input` is a built-in function that prompts the user to enter some input. The program waits for the user to enter the input, before executing the subsequent instructions. The value returned from this function is *always* a string. For example:

```
>>> input("What is your name? ")
What is your name? Jen
'Jen'
>>> name = input("What is your name? ")
What is your name? Jen
```

```
>>> name
'Jen'
>>> location = input("What is your location? ")
What is your location? Toronto
>>> location
'Toronto'
>>> print(name, "lives in", location)
Jen lives in Toronto
>>> num_coffee = input("How many cups of coffee? ")
How many cups of coffee? 2
'2'
```

Operations on strings

Operation	Description	Example	Output
`str1 + str2`	concatenate `str1` and `str1`	`print('ab' + 'c')`	`abc`
`str1 * int1`	concatenate `int1` copies of `str1`	`print('a' * 5)`	`aaaaa`
`int1 * str1`	concatenate `int1` copies of `str1`	`print(4 * 'bc')`	`bcbcbcbc`

Triple-quoted strings

We have used single- and double- quotes to represent strings. The third string format uses triple-quotes and a triple-quoted string cab span multiple lines. For example:

```
>>> print(''' How
are
you?''')
How
are
you?
```

Escape Sequences

Python has a special character called an *escape character*: \. When the escape character is used in a string, the character following the escape character is treated differently from normal. The escape character together with the character that follows it is an *escape sequence*. The table below contains some of Python's commonly used escape sequences.

Escape Sequence	Name	Example	Output
\n	newline (ASCII linefeed - LF)	print('''How are you?''')	How are you?
\t	tab (ASCII horizontal tab - TAB)	print('3\t4\t5')	3 4 5
\	backslash (\)	print('\')	\
\'	single quote (')	print('don\'t')	don't
\"	double quote (")	print("He says, \"hi\".")	He says, "hi".

Optional Book Reading

Chapter 4.3 Creating a Multiline String
Chapter 4.4 Printing Information
Chapter 4.5 Getting Information from the Keyboard

Video 4: Docstrings and Function help

Built-in function `help` displays the docstring from a function definition. For example, consider this function:

```
def area(base, height):
    """(number, number) -> number

Return the area of a triangle with dimensions base
and height.
"""

return base * height / 2
```

Calling `help` on function `area` produces this output:

```
>>> help(area)
Help on function area in module __main__:

area(base, height)
    (number, number) -> number

    Return the area of a triangle with dimensions base
    and height.
```

Optional Book Reading

3.3 Defining Our Own Functions

Video 5: Function Design Recipe

The Six Steps

1. Examples

- What should your function do?
- Type a couple of example calls.
- Pick a name (often a verb or verb phrase): What is a shor answer to "What does your function do"?

2. Type Contract

- What are the parameter types?
- What type of value is returned?

3. Header

- Pick meaningful parameter names.

4. Description

- Mention every parameter in your description.
- Describe the return value.

5. Body

- Write the body of your function.

6. Test

- Run the examples.

Applying the Design Recipe

The problem:

The United States measures temperature in Fahrenheit and Canada measures it in Celsius. When travelling between the two countries it helps to have a conversion function. Write a function that converts from Fahrenheit to Celsius.

1. Examples

```
>>> convert_to_ccelsius(32)
0
>>> convert_to_celsius(212)
100
```

2. Type Contract

```
(number) -> number
```

3. Header

```
def convert_to_celsius(fahrenheit):
```

4. Description

```
Return the number of Celsius degrees equivalent to fahrenheit degrees.
```

5. Body

```
    return (fahrenheit - 32) * 5 / 9
```

6. Test

```
 Run the examples.
```

Putting it all together:

```
def convert_to_celsius(fahrenheit):
    ''' (number) -> number

    Return the number of Celsius degrees equivalent to fahrenheit degrees.

    >>> convert_to_ccelsius(32)
    0
    >>> convert_to_celsius(212)
    100
    '''

    return (fahrenheit - 32) * 5 / 9
```

Optional Book Reading

Chaper 3.6 Designing New Functions: A Recipe

Video 6: Function Reuse

Calling functions within other function definitions

The problem: Calculate the semi-perimeter of a triangle.

The approach: Function `semiperimeter` calls function `perimeter`.

```
def perimeter(side1, side2, side3):
    '''(number, number, number) -> number

    Return the perimeter of a triangle with sides of length
    side1, side2 and side3.
    >>> perimeter(3, 4, 5)
    12
    >>> perimeter(10.5, 6, 9.3)
    25.8
    '''
    return side1 + side2 + side3


def semiperimeter(side1, side2, side3):
    '''(number, number, number) -> float

    Return the perimeter of a triangle with sides of
    length side1, side2 and side3.

    >>> semiperimeter(3, 4, 5)
    6.0
    >>> semiperimeter(10.5, 6, 9.3)
    12.9
    '''
    return perimeter(side1, side2, side3) / 2
```

Calling functions within other function calls

The problem: One triangle has a base of length 3.8 and a height of length 7.0. A second triangle has a base of length 3.5 and a height of length 6.8. Calculate which of two triangles' areas is biggest.

The approach: Pass calls to function `area` as arguments to built-in function max.

```
max(area(3.8, 7.0), area(3.5, 6.8))
```

Optional Book Reading

Chaper 3.1 Functions That Python Provides

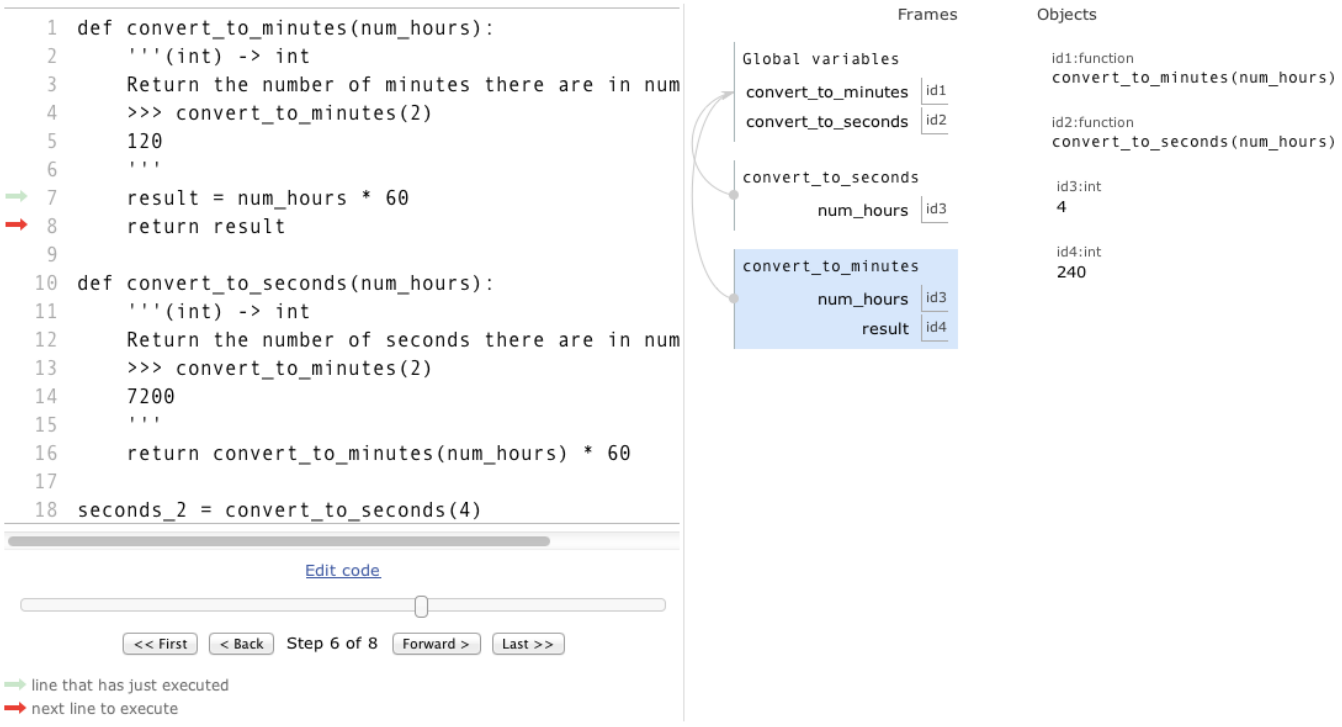
Video 7: Visualizing Function Calls

We can explore how Python manages function calls using the Python Visualizer. (See the Resources page.)

In the example below, function `convert_to_seconds` contains a call on `convert_to_minutes`.

```
def convert_to_minutes(num_hours):
    '''(int) -> int
     Return the number of minutes there are in num_hours hours.
     >>> convert_to_minutes(2)
        120
     '''
    result = num_hours * 60
    return result

def convert_to_seconds(num_hours):
    '''(int) -> int
    Return the number of seconds there are in num_hours hours.
    >>> convert_to_minutes(2)
    7200
    '''
    return convert_to_minutes(num_hours) * 60

seconds_2 = convert_to_seconds(4)
```

Here is what the memory model looks like just before the return statement inside function `convert_to_minutes` looks like:



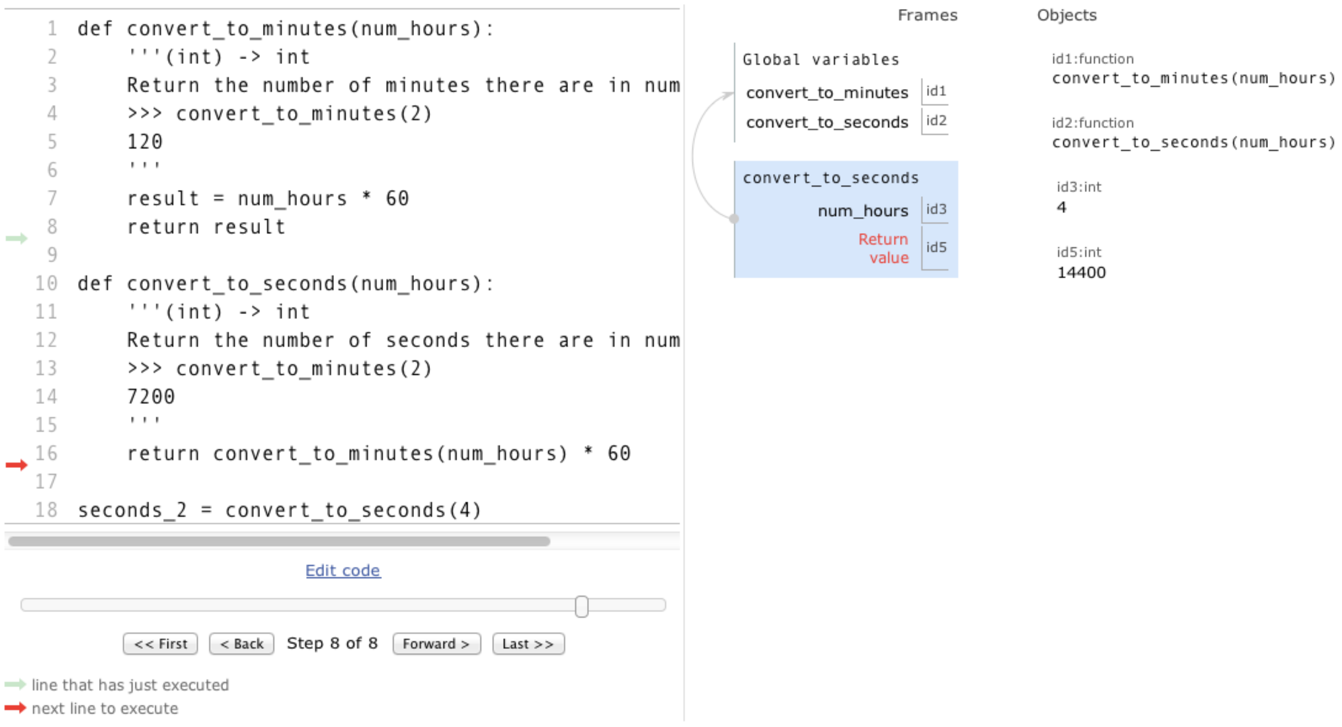
```
def convert_to_minutes(num_hours):
    '''(int) -> int
    Return the number of minutes there are in num
    >>> convert_to_minutes(2)
    120
    '''
    result = num_hours * 60
    return result

def convert_to_seconds(num_hours):
    '''(int) -> int
    Return the number of seconds there are in num
    >>> convert_to_minutes(2)
    7200
    '''
    return convert_to_minutes(num_hours) * 60

seconds_2 = convert_to_seconds(4)
```

```
Frames                                  Objects
Global variables
  convert_to_minutes  id1               id1:function
  convert_to_seconds  id2               convert_to_minutes(num_hours)
convert_to_seconds                      id2:function
  num_hours  id3                        convert_to_seconds(num_hours)
convert_to_minutes                      id3:int
  num_hours  id3                        4
  result     id4                        id4:int
                                        240
```

Note that there are three stack frames on the call stack: the main one, then underneath that a frame for the call on function `convert_to_seconds`, and underneath that the frame for the call on function `convert_to_minutes`.

Here is a link to the Python Visualizer at this stage of the execution so that you can explore this yourself. **We strongly encourage you to step backward and forward through this program until you understand every step of execution.**

When the return statement is executed, the call on `convert_to_minutes` exits. The bottom stack frame is removed, and execution resumes using the stack frame for `convert_to_seconds`:



```
def convert_to_minutes(num_hours):
    '''(int) -> int
    Return the number of minutes there are in num
    >>> convert_to_minutes(2)
    120
    '''
    result = num_hours * 60
    return result

def convert_to_seconds(num_hours):
    '''(int) -> int
    Return the number of seconds there are in num
    >>> convert_to_minutes(2)
    7200
    '''
    return convert_to_minutes(num_hours) * 60

seconds_2 = convert_to_seconds(4)
```

```
Frames                                  Objects
Global variables
  convert_to_minutes  id1               id1:function
  convert_to_seconds  id2               convert_to_minutes(num_hours)
convert_to_seconds                      id2:function
  num_hours     id3                     convert_to_seconds(num_hours)
  Return value  id5                     id3:int
                                        4
                                        id5:int
                                        14400
```

Optional reading

3.5 Tracing Function Calls in the Memory Model

Exercise shown in Python Visualizer (Start yourself)