CSE 115
Introduction to Computer Science I
Road map

▶︎ Review ◀

Defining functions

Calling functions

Function calls as expressions
Functions

Calling a function

The expressions 2*4 and 3-1 are 'arguments' of the function

\[ \text{pow}(2*4, 3-1) \] computes \(8^2\), or 64

the name of the function is 'pow'
Road map

Review

▶ Defining functions ◀

Calling functions

Function calls as expressions
Defining functions

We are not restricted to using built-in functions.

We can define our own!
Defining functions

Here's an example of a function definition:

def averageOfThree(x, y, z):
    average = (x + y + z) / 3
    return average
Defining functions

The parts are:

'def' is a keyword

```python
def averageOfThree(x, y, z):
    average = (x + y + z) / 3
    return average
```
Defining functions

'averageOfThree' is a name: you get to choose this

def averageOfThree(x, y, z):
    average = (x + y + z) / 3
    return average
Defining functions

'x', 'y', and 'z' are parameters.

A parameter is a variable whose value is assigned (from the corresponding argument) when the function is called.

def averageOfThree(x, y, z):
    average = (x + y + z) / 3
    return average
Defining functions

'(x, y, z)' is a parameter list.

The parentheses and commas are 'delimiters' (punctuation)

def averageOfThree(x, y, z):
    average = (x + y + z) / 3
    return average
Defining functions

```python
def averageOfThree(x, y, z):
    average = (x + y + z) / 3
    return average
```

: is also a delimiter
Defining functions

: is also a delimiter

It separates the header of the definition from its body

def averageOfThree(x, y, z):
    average = (x + y + z) / 3
    return average
Defining functions

The body is a block (sequence) of statements.

These statements are carried out when the function is called.

def averageOfThree(x, y, z):
    average = (x + y + z) / 3
    return average

Python refers to a block of statements as a "suite".
The return statement

The return statement consists of the keyword 'return' followed by an expression.

def averageOfThree(x, y, z):
    average = (x + y + z) / 3
    return average

The value of the expression average is returned as the value of the function.
The return statement

The value of the expression becomes the value of the function call.

def averageOfThree(x, y, z):
    average = (x + y + z) / 3
    return average
Road map

Review

Defining functions

Calling functions

Function calls as expressions
Calling functions

What happens when this function call takes place?

\[ \text{averageOfThree}(3, 7, 4) \]
Calling functions

**Step 1:** argument values are assigned to parameters

averageOfThree( 3, 7, 4 )
def averageOfThree( x, y, z ):
    average = (x + y + z) / 3
    return average
Calling functions

Step 1: argument values are assigned to parameters

averageOfThree(3, 7, 4)

def averageOfThree(x, y, z):
    average = (x + y + z) / 3
    return average

We keep track of the values of variables in a table.
Such a table is called an environment.
Calling functions

**Step 2:** statements in the body are executed in sequence

```python
def averageOfThree(x, y, z):
    average = (x + y + z) / 3
    return average
```

The body of a function is a sequence of statements.

<table>
<thead>
<tr>
<th>name</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>3</td>
</tr>
<tr>
<td>y</td>
<td>7</td>
</tr>
<tr>
<td>z</td>
<td>4</td>
</tr>
</tbody>
</table>
Calling functions

**Step 2a:** the assignment statement is done first

```
def averageOfThree( x, y, z ):
    average = (x + y + z) / 3
    return average
```

An assignment statement has the form:

```
variable = expression
```
Calling functions

**Step 2a:** the expression is evaluated

def averageOfThree(x, y, z):
    average = (x + y + z) / 3
    return average

To evaluate the expression, look up the values of all the variables in the environment table.
Calling functions

**Step 2a:** the expression is evaluated

```python
def averageOfThree(x, y, z):
    average = (x + y + z) / 3
    return average
```

<table>
<thead>
<tr>
<th>name</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>3</td>
</tr>
<tr>
<td>y</td>
<td>7</td>
</tr>
<tr>
<td>z</td>
<td>4</td>
</tr>
</tbody>
</table>

\[(x + y + z) / 3 = \]
Step 2a: the expression is evaluated

averageOfThree(3, 7, 4)

def averageOfThree(x, y, z):
    average = (x + y + z) / 3
    return average

(x + y + z) / 3 =
(3 + 7 + 4) / 3 =
Calling functions

**Step 2a:** the expression is evaluated

```python
def averageOfThree( x, y, z ):
    average = (x + y + z) / 3
    return average
```

<table>
<thead>
<tr>
<th>name</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>3</td>
</tr>
<tr>
<td>y</td>
<td>7</td>
</tr>
<tr>
<td>z</td>
<td>4</td>
</tr>
</tbody>
</table>

\[(x + y + z) / 3 = (3 + 7 + 4) / 3 = 14 / 3 = 4.666666666666667\]
Step 2a: the variable is assigned the value of the expression

averageOfThree( 3, 7, 4 )

def averageOfThree( x, y, z ):
    average = (x + y + z) / 3
    return average

(x + y + z) / 3 =
(3 + 7 + 4) / 3 =
14 / 3 = 4.666666666666667
Calling functions

**Step 2a:** the variable is assigned the value of the expression

```python
def averageOfThree(x, y, z):
    average = (x + y + z) / 3
    return average
```

<table>
<thead>
<tr>
<th>name</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>3</td>
</tr>
<tr>
<td>y</td>
<td>7</td>
</tr>
<tr>
<td>z</td>
<td>4</td>
</tr>
<tr>
<td>average</td>
<td>4.666666666666667</td>
</tr>
</tbody>
</table>

\[
(x + y + z) / 3 = 14 / 3 = \approx 4.666666666666667
\]
Calling functions

### Step 2b: the return statement is executed next

```python
def averageOfThree(x, y, z):
    average = (x + y + z) / 3
    return average
```

A return statement has the form:

```
return expression
```

<table>
<thead>
<tr>
<th>name</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>3</td>
</tr>
<tr>
<td>y</td>
<td>7</td>
</tr>
<tr>
<td>z</td>
<td>4</td>
</tr>
<tr>
<td>average</td>
<td>4.666666666666667</td>
</tr>
</tbody>
</table>
Calling functions

**Step 2b:** the expression is evaluated

averageOfThree( 3, 7, 4 )

def averageOfThree( x, y, z ):
    average = (x + y + z) / 3
    return average

<table>
<thead>
<tr>
<th>name</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>3</td>
</tr>
<tr>
<td>y</td>
<td>7</td>
</tr>
<tr>
<td>z</td>
<td>4</td>
</tr>
<tr>
<td>average</td>
<td>4.666666666666667</td>
</tr>
</tbody>
</table>

The expression is simple: the value of a variable expression is simply the value associated with the variable in the current environment table.
Calling functions

**Step 2b:** the expression is evaluated

```python
def averageOfThree(x, y, z):
    average = (x + y + z) / 3
    return average
```

<table>
<thead>
<tr>
<th>name</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>3</td>
</tr>
<tr>
<td>y</td>
<td>7</td>
</tr>
<tr>
<td>z</td>
<td>4</td>
</tr>
<tr>
<td>average</td>
<td>4.666666666666667</td>
</tr>
</tbody>
</table>
Calling functions

**Step 2b:** the expression is evaluated

```python
def averageOfThree( x, y, z ):
    average = (x + y + z) / 3
    return average
```

<table>
<thead>
<tr>
<th>name</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>3</td>
</tr>
<tr>
<td>y</td>
<td>7</td>
</tr>
<tr>
<td>z</td>
<td>4</td>
</tr>
<tr>
<td>average</td>
<td>4.666666666666667</td>
</tr>
</tbody>
</table>

average = 4.6666666666666667
Road map

Review

Defining functions

Calling functions

▶ Function calls as expressions ▷
Step 3: the function call is an expression…

averageOfThree( 3, 7, 4 )
Step 3: ...whose value is the value returned by the function

\[
\text{averageOfThree}(3, 7, 4) = 4.666666666666667
\]
Calling functions

Another example: the value of

averageOfThree( 5, 2, 8 )
Calling functions

Another example: the value of

\[
\text{averageOfThree}( 5, 2, 8 )
\]

```python
def averageOfThree( x, y, z ):
    average = (x + y + z) / 3
    return average
```

<table>
<thead>
<tr>
<th>name</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>5</td>
</tr>
<tr>
<td>y</td>
<td>2</td>
</tr>
<tr>
<td>z</td>
<td>8</td>
</tr>
</tbody>
</table>
Calling functions

Another example: the value of

\[ \text{averageOfThree}( 5, 2, 8 ) \]

def averageOfThree( x, y, z ):
    average = (x + y + z) / 3
    return average

\[(x + y + z) / 3 = (5 + 2 + 8) / 3 = 15 / 3 = 5.0\]
Calling functions

Another example: the value of

\[
\text{averageOfThree}( 5, 2, 8 )
\]

```python
def averageOfThree( x, y, z ):
    average = (x + y + z) / 3
    return average
```

The value returned is 5.0
Calling functions

Another example: the value of

averageOfThree( 5, 2, 8 )

is 5.0