CSCI 141

Scott Wehrwein

Program Execution: Statements and Expressions
Function Calls: Return Values

Goals

- Understand the distinction between a statement and an expression

- Understand function calls as expressions that evaluate to their return values

Statements and Expressions

- A statement is a line (or multiple lines) of code that Python can execute.

`my_name = “Scott”` is an assignment statement

A statement in Python does not evaluate to a value!

- An expression is a combination of values, variables, operators, and function calls that Python evaluates to determine its **value**.

```
type(32)
2+2
int(a)
int(b) * 4
```

are all expressions

The notation => is often used to mean “evaluates to”:

```
2 + 2 => 4
```

“two plus two evaluates to four”

Note: => is **not** a Python operator

Function Calls, Revisited

- Recall: function can take inputs called arguments `int(7.9)`
- New: A function can give back an output, called its return value. `my_val = int(7.9)`
- A function call is an expression that evaluates to its return value.

Some functions return values

Examples:

`int` returns an `int`

`float` returns a `float`

`str` returns a `str`

`int(4.6)` returns `4`

`str(4.6)` returns `"4.6"`

`input` returns a `str`

`input` returns whatever text the user entered

`print` does not return a value

if used as an expression, it evaluates to `None`

`None` is a special keyword meaning no value

A function call *evaluates to* its return values

Examples:

`float(int(6.8))` evaluates to `6.0`

because

`int(6.8)` evaluates to `6`

`float(6)` evaluates to `6.0`

```
name = input("Enter your name")
```

stores whatever the user typed in the variable `name`

Note: `input` always returns a `str`

Beware!

`input` always returns `a str`

Implication:

```
# ask for a number
a = input("Enter a number: ")
# but a is a string, so we need to:
user_number = float(a)
# now user_number has type float

# we can do it in one line:
a = float(input("Enter a number:"))
```

Putting it all together

- Consider this program:

```
a = 4
b = float(2 + a)
```

- What happens when we execute it?

Putting it all together

- Consider this program:

```
a = 4

b = float(2 + a)
```

- What happens when we execute it?
  - the value `4` gets stored in `a`

Putting it all together

- Consider this program:

```
a = 4
b = float(2 + a)
```

- What happens when we execute it?
  - the value `4` gets stored in `a`
  - the expression `2+a` is evaluated, resulting in the value 6

Putting it all together

- Consider this program:

```
a = 4
b = float(6)
```

- What happens when we execute it?
  - the value `4` gets stored in `a`
  - the expression `2+a` is evaluated, resulting in the value 6

Putting it all together

- Consider this program:

```
a = 4
b = float(6)
```

- What happens when we execute it?
  - the value `4` gets stored in `a`
  - the expression `2+a` is evaluated, resulting in the value 6
  - `6` is passed into the `float` function

Putting it all together

- Consider this program:

```
a = 4
b = 6.0
```

- What happens when we execute it?
  - the value `4` gets stored in `a`
  - the expression `2+a` is evaluated, resulting in the value 6
  - `6` is passed into the `float` function
  - the `float` function converts 6 to a `float` and returns `6.0`

Putting it all together

- Consider this program:

```
a = 4
b = 6.0
```

- What happens when we execute it?
  - the value `4` gets stored in `a`
  - the expression `2+a` is evaluated, resulting in the value 6
  - `6` is passed into the `float` function
  - the `float` function converts 6 to a `float` and returns `6.0`
  - the value `6.0` gets stored in variable `b`

Putting it all together

- In what order do things get evaluated?
- A function's arguments are always evaluated left-to-right before it is called:

```
print(2+2, 4+6, int(10.4))
print(4, 4+6, int(10.4))
print(4, 10, int(10.4))
print(4, 10, 10)
```

`4 10 10` is printed to the console

Demo

Demo

- storing input's return value in a variable and converting its type

- function call with no return value (e.g., `print`)

- The Thonny Shell is a REPL (read-evaluate-print loop).

  - An expression on its own line in a program vs expression in the Thonny shell