

More symbol tables, scope, conditionals

Oliver W. Layton

CS151: Computational Thinking: Visual Media

Lecture 07, Fall 2020

Wednesday September 9

Review: Assignment symbol table

```
def convertFtoC(temp):  
    convertedTemp = (5/9)*(temp - 32)  
    print(temp, 'degrees in F is', convertedTemp, 'degrees in C.')
```

Main code

```
tempInF1 = 32  
convertFtoC(tempInF1)
```



```
tempInF2 = 212  
convertFtoC(tempInF2)  
tempInF1 = 100
```

One more thing we now add to symbol tables: modules

Let's draw symbol table for the following

```
import turtle  
turtle.forward(10)  
turtle.forward(20)
```

Scope

Local scope: the variables and functions only may be accessed or have local scope inside their respective current symbol table.

This is why we can't access `convertedTemp` from main code.

Flavors of function parameters

There are two types: **positional arguments** and **keyword arguments**.

Positional function arguments

In our temperature converter program, the function parameter `temp` in `convertFtoC(temp)` is a **positional argument**:

```
def convertFtoC(temp):  
    convertedTemp = (5/9)*(temp - 32)  
    # blah blah blah
```

In general we can have lots of positional arguments as in `myFun(a, b, c, d, e)`,

- You are **REQUIRED** to specify **ALL** positional arguments in order to call the function.

Keyword function arguments

You have the option to omit when calling the function. If you do, the function uses a **default value** instead:

```
def hexagon(x, y, s, penWidth=5, interiorColor='yellow'):  
    # code here
```

- Valid call: `hexagon(0, 0, 1)`. Keyword arguments are **optional**.
- We can **override** a default value by just passing in a new value by name: `hexagon(0, 0, 1, interiorColor='red')`
- We can override as many or as few keyword arguments as we like.

```
def hexagon(x, y, s, penWidth=5, interiorColor='yellow'):
    # lots of code in here
```

These calls won't work:

```
# Invalid because a positional argument cannot follow an optional argument
hexagon(300, 300, 0.5, penWidth=5, 3)
# Why doesn't this work?
hexagon(300, 300, penWidth=5)
```

These do work:

```
# Keyword arguments order can be swapped
hexagon(300, 300, 0.5, interiorColor='yellow', penWidth=5)
# You can use keywords to refer to positional arguments
hexagon(x=300, y=300, s=0.5)
```

Argument rules

1. All positional arguments must be provided when you call a function.
2. No positional arguments after keyword arguments
3. Keyword arguments can be specified in any order.
4. Positional arguments can be referred to by name, like keyword arguments.

Conditional statements

New type of statement to allow us to prepare for the unexpected!

if statement

```
if [Boolean condition statement]:  
    [body]
```

If there's traffic, then we want to take a different route. If it's raining, go outside with an umbrella. If we're drawing a house, we'll want to draw a rectangle and triangle.

The condition **MUST** evaluate to True to enter the `if` statement body (indented).

Note the :

Example if statement

```
if i_am_happy:  
    print(':)')
```

```
print('hi 1')
```

```
if i_am_sad:  
    print(':(')
```

```
print('hi 2')
```

- Code executes line-by-line, top-to-bottom like usual. `i_am_happy` and `i_am_sad` variables MUST be either `True` or `False` (boolean type).
- `print(':)')` only run IF `i_am_happy` is assigned the value `True` (otherwise `print('hi 1')` is run next after Line 1).

`if else` statement

After the body of the `if` statement, you have the option of putting an `else` statement (not required), and Python will evaluate code in the body of the `else`, assuming the Boolean condition evaluates to `False`:

```
if [Boolean condition statement]:  
    [body]  
else:  
    [body]
```

- Python *skips over the body of the* `else`, if it enters the `if` clause, which makes sense.

Example if else

```
if i_am_happy:  
    print(':)')  
else:  
    print(':( ' )  
  
print('hi')
```

- if `i_am_happy` is `True`, `print(':)')` run then `print('hi')`.
- if `i_am_happy` is `False`, `print(':( ' )` run then `print('hi')`.

Adding more conditions to check with `elif` clauses

Short for `else if`.

```
if isSnowing:  
    getSkiJacket()  
elif isRaining:  
    getRainJacket()  
else:  
    getSunGlasses()
```

`isSnowing` and `isRaining` are booleans.

Relational operators

Operator	Definition
<	strictly less than
<=	less than or equal to
>	strictly greater than
>=	greater than or equal to
==	equal to
!=	not equal to

- Examples: `x == y`, `x != 10`, `4 > 2`.
- All of these will evaluate to `True` or `False` (a Boolean value).
- These operators have lower precedence than math operators (which gets evaluated first?) and equal precedence amongst themselves.

Let's work on Question 1 of the
conditional worksheet