CS 106
Introduction to Computer Science I

07 / 21 / 2021

Instructor: Michael Eckmann

Today's Topics

- Questions / comments?
- Some odds and ends
  - Function definitions with optional parameters and default values
  - else can be used with for and while loops
  - try/except (to handle raised exceptions)
  - continue statement in loops
  - list comprehensions
  - In class exercise together

Functions w/ optional parameters

- In the list of parameters you specify within ( ) you may specify required parameters (what we've done so far), but also after any required parameters we can have optional parameters (these have a default value)
- e.g.

```
def example_fun(req1, req2, opt1='Hello', opt2=88):
        # code for the function here
```

Functions w/ optional parameters

- Let's write a function that uses optional parameters to say generate a random number defaults from 1 to 10, but if the caller can optionally pass in different values.
- Let's write a function that has 1 required parameter and an optional parameter (e.g. a name and a greeting)

```
def greet(name,greeting='Hello'):
    print(greeting, name)
```

else after a loop

- For loops and while loops allow an optional else clause after them that executes after a loop finishes iterating (but is not executed if the loop ended due to a break)
- So, it allows you to put code inside the else to be executed only if the loop terminated due to the while condition becoming False, or all the iterations of the for loop completed (as opposed to those loops stopping due to a break statement)

try/except/else/finally

- Try except blocks allow us to “catch” a raised exception and have the program execute code when an exception is raised in the try rather than crash.
- Else is optional --- contains code that will execute when the exception is not raised by the code in the try.
- Finally is optional --- contains code that will execute after the try/except regardless of whether an exception was raised
- In the except we can explicitly catch named exceptions or catch any exceptions
- Can have multiple excepts to catch different exceptions

try/except/else/finally

- Let's write a function that uses try/except to get an int from the user, but continually asks for an int if they do not type on (must use a loop around it)

continue statement in loops

- The continue statement is used in loops to
  - Not execute the rest of the code for the current iteration
  - But then continue with the next iteration

- Contrast this with break
  - The break statement terminates the loop (stops the current and future iterations)
- As an example for us to use continue, let's write code that sums up all the numbers from 1 to 10 excluding 3,5 and 7

List comprehensions

- list comprehensions
- code to build a list like the following:

```
nums1 = []
for n in range(1,200):
    nums1.append(n ** 2 + 4*n + 6)
```

can be replaced by:

```
nums = [n ** 2 + 4*n + 6 for n in range(1,200)]
```

List comprehensions

- filtered list comprehensions
- code to build a list like the following:

```
odd_nums1 = []
for n in nums1:
    if n % 2 == 1:
        odd_nums1.append(n)
```

can be replaced by:

```
odd_nums = [n for n in nums if n %2 == 1]
```

Let's write a larger program together

- Let's use the cards dictionary from yesterday and we'll write a complete blackjack program that deals random blackjack hands and tells the user the result of them and allows the user to hit or stay
- We'll write it so that the user can continually play hands of blackjack until they indicate they are done.
- If there's time, we'll add code to play they're blackjack hand against the computer's hand as well
- I'll jot down ideas on the output and input expected on the board

Let's write a larger program together

- Let's write at least three functions and code that executes them
  - A display_hand function
  - A function to compute the value of a hand
  - A function to deal a hand (including asking user to hit or stay)