CS1210 Lecture 5   Sep. 1, 2021

- HW1 due Friday
- DS1 Assignment mostly graded - all will be graded by 5pm tomorrow, perhaps today
  - hawkID() should not return your student ID *number*
  - Submissions - DS and HW - should contain *only* defs. No "testing" code that runs automatically at the top level!
  - Submissions should not containing any print statements not specifically requested (e.g. not prints used for debugging left in the code)
  - Make it a habit to check the ICON comments on your assignment even if you get full credit. Sometimes we have something to say that could be relevant for the next assignment
- Last time
  - Math functions and the math module
  - Chapter 6: function calls and defining new functions
  - HW1 information and advice
- Today
  - BeginningCh 7: logical/Boolean expressions, conditional execution – if / else
  - More Ch 6 on functions
    - Execution flow of control
    - Variables and parameters are local
    - Stack diagrams
    - return vs. print
  - More on HW1
    - Floor division (//) and modulus (%) operators (from Ch 2.7)
    - Math.ceil() function
    - String format, f strings, and controlling number of printed decimal places
    - Fruitful functions, incl return vs. print

From last time: Ch 6 - Defining New Functions

```
def foo(a, b, c):
    temp = a * b
    result = temp + c
    return result
```

IMPORTANT

When executing a function call:

1) first, the function's parameter variables are bound to the *values* of the function call's arguments

2) then, body of the function is executed

```
>>> x = 3
>>> foo(x * x, x + 1, 3) ←
```

foo will be executed with variable a bound to 9, b bound to 4, c bound to 3

foo "knows" nothing about x. x *isn't* passed in. 9, 4, and 3 are passed into foo.

Ch 6: Defining New Functions

Functions can have zero parameters:

```
def giveBack1():
    return(1)


>>> giveBack1()
1
```

You can type the function name at interpreter prompt:

```
>>> giveBack1
<function giveBack1 at 0x10367cb70>
>>> type(giveBack1)
<class 'function'>
```

Note: a function name is just like any other variable. It is bound to a value; in this case that value is a function object. Don't try to use that same name as a simple variable (e.g. giveBack1 = 3). You'll "lose" your function. You can even "break" Python – it doesn't prevent you from redefining built-in things:

```
>>> len("hello")   # len is an important standard Python function!
5
>>> len = 23
>>> len("hello")
 ERROR
```

Before looking at functions in more depths, we'll skip to Ch 7: Conditional execution

- Recall list of key programming components: Expressions, variables and assignment, functions, if-then-else (conditional execution), iteration

- We've covered three of them – expressions, variables and assignment, the basics of functions. Can't easily compute a lot yet. Need the last two – conditional execution and iteration.

- Before looking at functions in more depth (more Ch 6), we'll skip to basic "conditional execution" from Ch7 so that I can use if/else in examples

Ch7.1-7.3: Boolean expressions and Logical Operators

- Boolean expressions: have True/False as value

  E.g. 3 == (5 – x).

- Boolean/relational operators:

  <, >, >=, <=, !=

  E.g. (x + y) < 4

- Logical operators

  and, or , not

  E.g. (x < 5) or (z == "foo")

Understand these well. Critical for conditional execution, decision making in programs!

Ch7.4-7.7: Condition execution - **if**

The most basic conditional statement is **if**

      if (Boolean expression):

        …

        …   lines of code that execute when Boolean

        …   expression evaluates to True

…

… more lines of code. These execute whether or

… not Boolean expression evaluated to True

…

Ch7: Condition execution – **if**-**else**

if (Boolean expression):

    …

    …   code that executes when Boolean expr

    …   evaluates to True

else:

    …

    …   code that executes when Boolean not true

    …

…

… code that executes after if-else statement, whether

… Boolean expression was True or not

…

Ch7: **if-else**

print('before if')
if (a < b):
    print('a is smaller')
else:
    print('a is not smaller').
print('after if')
print('goodbye')

NOTE: indentation semantically important. Has execution effect.

Return to Ch 6/Functions: Control Functions

A program consists of a list of instructions, executed (except in certain situations) in top-to-bottom order

x = 3   ←1   def foo(a, b, c):
y = x + 13   ←2   temp = a + (b * c)   ← 4
z = foo(x, y, 3)   ← 3   temp = temp/3.0   ← 5
newVal = math.sqrt(z)   ←7   return temp   ← 6
result = x + newVal   ←8
print result   ←9

As part of 3, foo parameter
a is bound to 3 (not x!)
b is bound to 16 (not y!)
c is bound to 4

Ch 6.4: Variables and parameters are local

- A function's parameters are **local** variables. That is, they exist only during the execution of that function.

- Other variables that are assigned-to within a function body are also local (note: we'll see exception to this later)

- Top-level variables (not within **def**s) are called **global**

```
def foo(a, b):
    c = (a + b) * 2
    return(c)
```

```
>>> foo(3, 4)
    14
>>> a
    Error
>>> c
    Error
```

IMPORTANT!

Ch6: local/global variables

- Another way to say this is that each function has its own **scope** or **namespace**

```
>>> def f(x):
        y = 1
        x = x + y
        print(x)
        return x


>>> x = 3
>>> y = 2
>>> z = f(x)
4
>>> x
?    3
>>> y
?    2
>>> z
     4
?
```

Function f's variable x. A **local** variable that exists only within scope of definition of f

Two completely different variables!

"Top-level" or **global** variable x

Hint: think of them as, e.g., x_f and x_{global}

Ch6: Stack frames

- At top level (the interpreter/shell) a "symbol table" keeps track of all variables (and values bound to them) defined at that level

- When a function is called, a new symbol table , or **stack frame**, is created to keep track of variables defined in the function and the values associated with those names. This includes (1) the function's parameters and (2) any other variables assigned to within the function

  – When the function finishes, the stack frame goes away.

- *Note: this isn't just one level, of course. As functions call other functions that call other functions, the stack (of stack frames) grows deeper …*

Ch6: Stack frames

```
>>> def f(x):
        y = 1
        x = x + y
        print(x)
        return x

>>> x = 3
>>> y = 2
>>> z = f(x)
4
>>> x
>>> y
>>> z
```

f: x ~~3~~ 4

          y 1

main: x 3

          y 2

          z 4

Will demo/discuss more complex
example Friday

return vs print

print and **return** are very different (and some students never quite get this!)

```
def add1a(number):
    result = number + 1
    return result


def add1b(number):
    result = number + 1
    print(result)
```

Explore difference by using invoking them in shell in a couple of ways.

Ch 6: Why functions?

- Functions let you name a group of statements that make up a meaningful computation.
- Make programs easier to read and debug
  - E.g. debug functions one by one, assemble into whole program
- Make programs shorter (in lines of code) by eliminating duplication. If you start seeing same (or nearly the same) segments of code in various places in your program, consider defining a function and replacing the common segment with function calls.
- Well designed functions often useful in other programs you write.

HW1 advice/hints

- Take baby steps:
  - write a line or two
  - test
  - repeat!
- While debugging, use lots of print statements!
  - Sometimes even worth printing values after every line of code!
  - (remember to comment out/remove such debugging statements before submitting your homework)
- Consider using math.ceil() function for computing nights needed
  - I use it to get integer number of full 8-hour "days" needed and also integer number of full 40-hour "weeks" needed (useful for computing number of rest days needed)
- Some people have asked me about lastDayFraction in my HW1 suggested outline. There are many ways to do things. That's just how I think about it. Once the number of nights needed is known, what costs can we compute?

  - Hotel costs breakfast cost, dinner cost ✅

  - Lunch cost ❌
  - I calculate a value lastDayFraction to enable lunch cost calculation
- Use GOOD conversion factors. It is easy to find good ones.
  - Google "miles kilometers international standard" …
    - 1.609344
  - Google "gallons liters international standard"
    - 3.785411784

Formatting strings and outputting numbers with fixed number of decimal digits

- https://docs.python.org/3/tutorial/inputoutput.html
- string **format** method and "f-strings" make it easy to print nice complicated strings and to control number of decimal places in printed numbers

  ```
  >>> item = "bicycle"
  >>> itemCost = 127.123124
  >>> outputString = "The {}'s cost is S{:.2f}.".format(item, itemCost)
  >>> print(outputString)
  The bicycle's cost is $127.12.

  OR

  >>> outputString = f"The {item}'s cost is ${itemCost:.2f}."
  >>> print(outputString)
  The bicycle's cost is $127.12.
  ```

Floor Division and Modulus (Ch 2.7) and math.ceil function

- //: "floor division operator. Divides two numbers and truncates (rounds down) to whole number

  14 // 3 yields 4

  8.1 // 2 yields 4.0

  % very useful for, e.g., odd/even tests

- %: modulus operator yields remainder

  11 % 3 yields 2

  ((14 // 3) * 3) + (11 % 3) == 14 and in general (except for possible floating point issues), ((n // d) * d) + (n % d) == n

- math.ceil(x) yield nearest integer equal or greater than x. E.g. math.ceil(23.01) -> 24 and math.ceil(16.0) -> 16

  Hint: I use this as key part of calculating required hotel nights

HW1: function computeTripData

def computeTripData (distanceK, vehSpeedMPS, vehKPL, gasCostPerLiter, breakfastCostPerDay, lunchCostPerDay, dinnerCostPerDay, hotelCostPerNight) :

    ....

    .... Lines of code that calculate, in terms of parameters, cost of trip

    ....

    return tripHours, gasCost, tripCost, numBreakfasts,
                            numLunches, numDinners, numHotelNights

To the *user* of function, it's a "black box." User sends in values, sees printed answer!

HW1: function computeTripData

```python
def computeTripData(distanceK, vehSpeedMPS, vehKPL, gasCostPerLiter,
              breakfastCostPerDay, lunchCostPerDay, dinnerCostPerDay, hotelCostPerNight) :
    tripCost = 0
    # compute driving time of trip in hours
    tripHours = …
    # compute liters of gas needed
    litersNeeded = …
    # compute gas cost
    gasCost = …

    # compute number of nights (hotel stays) needed.
    # This is a little bit tricky. To get it, you need to determine both the number of full 8-hour days
    #  driven and also determine the number of rest days needed.
    …
    hotelNightsNeed = ….              This must be an integer!

    # perhaps compute a number (0.0->1) representing fraction of final day.
    lastDayFraction = …

    # compute breakfast, lunch, and dinner costs
    breakfastCost = …
    # determining lunchCost might involve more than one line and an if statement that uses the lastDayFraction
    …
    dinnerCost = …

    # sum costs
    tripCost = …

    # return results
    return tripHours, gasCost, tripCost, … …., hotelNightsNeeded
```

HW1: printTripSummary

```
def printTripSummary(vehName, distanceM, …) :

    # assign some new variables values based on converting units in input
    #
    distanceKM = kilometersFromMiles(distanceM)        # if you copy the function from DS1

    ….


    # make a function call to computeTripData, passing in values that are in the
    # units expected by computeTripData
    #
    tripHours, gasCost … , hotelNights = computeTripData(distanceKM, …)


    # print and return summary information
    #
    tripSummaryString = ….
    print(…)
    return …
```

HW1 - test functions?

- In prior years, on assignments like this one I included requirements like "Write function testQ1() that makes at least five calls to tripCostData(...) with different arguments."
- Students email me – "what's this, dude????"
- You should get in the habit of writing functions that you use to test your code as you are debugging it.
    - When you write computeTripData you probably (I hope!) test it by trying
      >>> computeTripData(....)
      multiple times with different values in the interpreter. But that can get tedious, especially for larger, more complicated program.

      It can be better to "package up" the tests, so you can easily run them.
    -
        def testQ1():
          computeTripData(....)
          computeTripData(... different arguments...)
          computeTripData(... different arguments...)

      NOTE: Something is not right here! What?
      tripCostData doesn't print anything so
      testQ1() will run but show you nothing
      FIX? *Use print(tripCostData(...))*

          Then, every time you change computeTripData, you just run testQ1 and examine the output versus what it should be.
- It's important that you choose the arguments of your tests carefully! Think about what sets of argument test the variety of situations that the function should handle
- Similarly, for Q2:
    - def testQ2():
        printTripSummary(…)   *NOTE: unlike for testQ1(), no print(…)*
        printTripSummary(…)   *needed around printTripSummary()*
        printTripSummary(…)   *because it already*
                                     prints information

HW1 - test functions

- It's important that you choose the arguments of your tests carefully! Think about what sets of argument test the variety of situations that the function should handle

- def testQ1()

  print(computeTripData(100.0, 100.0, 1.0, 1.0, 0.0, 0.0, 0.0, 0.0))
  print(computeTripData(200.0, 100.0, 1.0, 1.0, 0.0, 0.0, 0.0, 0.0))

  …

- from random import random

  def testq1()
    print(computeTripData(random(),random(),random(),random(),random(),random(), random(), random()))
    print(computeTripData(random(),random(),random(),random(),random(),random(), random(), random()))
    print(computeTripData(random(),random(),random(),random(),random(),random(), random(), random()))
    print(computeTripData(random(),random(),random(),random(),random(),random(), random(), random()))
    print(tcomputeTripData(random(),random(),random(),random(),random(),random(), random(), random()))

  Not effective for testing!!
        - for the random case, would you know the   expected
  results? It's not a good test if you don't   know expected results
  to compare to actual results

A few words on debugging and testing

- For beginning programmers, I'm not a big advocate of using "real" debuggers (the tools that come with IDEs like Wing and even IDLE).

- I'm a fan of *reading code carefully and critically* and
  - Using a lot of carefully placed print statements – print early, print often - see where things go wrong!
    - in HW1, if printed final results don't seem right, print out all your intermediate variables. Most people have variables for, e.g. driving time, gas needed, gas cost, hotel nights needed, etc. – print them all!

- I am also a fan of "baby steps": as I said earlier, write a little bit of code, then test, then a little more, and test again, etc. Small changes to a so-far-working program makes errors much easier to find.

- More to say on this later …

Next time

- Finish conditional execution part of Chapter 7
  - nested conditionals, if-elif-else

- start iteration (the last of the five key programming components). Read Ch 8 (it refers back to introduction to "for" in 4.4 but you don't really need to look at 4.4)