Lesson 10:

1. How to Solve Problems

Important skill.

1. Days Between Dates

I already did this, but I used the datetime library.

1. Quiz: First Step

Make sure we understand the problem.

1. Understanding a Problem

Computational Problem, need to know possible inputs.

1. The First Rule

Pythonista’s Guide to All Problems in the Galaxy

1. Don’t Panic!
2. What are the Inputs?
3. What Are the Inputs

Inputs: two dates

Defensive programming = make sure the second date is not before the first date.

Gregorian Calendar

1. How Are Inputs Represented

def daysBetweenDates(year1, month1, day1, year2, month2, day2)

might use a package for inputs

1. Quiz: What Are the Outputs

Return a number giving the number of days between the first date and the second date.

1. Obey the Rules

Pythonista’s Guide to All Problems in the Galaxy

1. What are the outputs?
2. Solve the problem.
3. Quiz: Next Step

Work out some examples to find test cases for the problem.

1. Quiz: The Expected Output

0, 1, undefined, 365, undefined

1. Take the Next Step

Try examples.

1. Try an Example

Count remainder of January, all of the days in the months between, and then the number of days in June.

1. Harder Example

Take years into account.

1. Algorithm Pseudocode

Try to get the idea down and see if we can solve it.

days = # days in current month – day1

month1 += 1

while current month<target month:

add days of month until current month = target month

add days of the year until year is equal to target year

1. Quiz: Should We Implement It

We should try to find a simpler way of solving the problem.

Computers are not lazy, a simple mechanical way is good.

1. Different Approach

While date1 is before day 2, advance to the next day.

1. Quiz: Simple Mechanical Algorithm

A few milliseconds. ~36500 days to count, each day = 1000 instructions

36.5M instructions = 0.036 seconds

1. Don't Optimize Prematurely

Not worth the effort since the program is fast enough for its use case.

1. Quiz: What Should We Write First

nextDay(year, month, day) solves for the simplest case

1. Quiz: Define Simple nextDay

def nextDay(year, month, day):

"""

Returns the year, month, day of the next day.

Simple version: assume every month has 30 days.

"""

if month == 12 and day == 30:

year += 1

month = 1

day = 1

elif day == 30:

month += 1

day = 1

else:

day += 1

return year, month, day

1. Making Progress Is Good

We should be happy making progress, it is a good thing!

1. Quiz: What Should We Do Next

I like the beach, but define daysBetweenDates to give approximate answers to our problem.

1. Quiz: Define daysBetweenDates

def daysBetweenDates(year1, month1, day1, year2, month2, day2):

"""Returns the number of days between year1/month1/day1

and year2/month2/day2. Assumes inputs are valid dates

in Gregorian calendar, and the first date is not after

the second."""

count\_days = 0

while year1 < year2 or month1 < month2 or day1 < day2:

year1, month1, day1 = nextDay(year1, month1, day1)

count\_days += 1

return count\_days

1. Step One Pseudocode
2. Step Two Helper Function

Made a helper function to check is the first date is less than the second date.

1. Step Three daysBetweenDates
2. Quiz: Test for Valid Inputs

assert not dateIsBefore(year2, month2, day2, year1, month1, day1)

1. Real World Problem

Not all months have 30 days

1. Quiz: Best Strategy
2. Quiz: Finish daysBetweenDates

def nextDay(year, month, day):

"""Modified version: uses daysInMonth"""

if day < daysInMonth(year, month):

return year, month, day + 1

else:

if month == 12:

return year + 1, 1, 1

else:

return year, month + 1, 1

def daysInMonth(year, month):

days\_each\_month = [31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31]

if isLeapYear(year):

days\_each\_month[1] = 29

return days\_each\_month[month - 1]

def isLeapYear(year):

if year % 4 != 0 or (year % 100 == 0 and year % 400 != 0):

return False

1. Solution Step I
2. Solution Step II
3. Solution Step III
4. Solution Step IV
5. Conclusion

Pythonista’s Guide to All Problems in the Galaxy

1. Simple mechanical solution
2. Develop incrementally and test as you go