Recursion (part 2)

November 10, 2021

Administrative notes

Project 2

Exam 2

Recursion

The Fibonacci sequence, again

1, 1, 2, 3, 5, 8, 13,...

After the first two numbers, each number is the sum of the previous two numbers

That is, f(n) = f(n-1) + f(n-2)

Iterative

```
def fib(n):
   If n<= 3:
        return n
   Else:
        fib = [1,1]
        for i in range(2,n+1)
            fib.append(fib[i-1] + fib[i-2])
        return (fib[n])
```

Recursive

```
def fib(n):
   if n < 3:
      return 1
   else:
      return (fib(n-1) + fib(n-2))
```

Another visualization of recursion

https://recursion.vercel.app/

How do you solve a problem using recursion?

1. What is the base case? #there may be more than one
2. How do I describe a subproblem of my problem
    a. If I repeated making that subproblem, do I get a base case?
    b. At this point we should TRUST the recursive calls
3. Assuming I have the solution to the subproblem, ***HOW DO I SOLVE MY PROBLEM WITH IT?***
    a. I should be careful to make sure I'm returning the answer to MY PROBLEM

Palindromes

Calculate whether a string is a palindrome using recursion

What's the base case?

- A string that is zero characters long - an empty string - IS a palindrome
- A string that is one character long IS a palindrome
- A string where the first character is DIFFERENT from the last character is NOT a palindrome
    - "cat" is NOT a palindrome
-

Palindromes - recursive case

What about the recursive case?

- IF the first character is the SAME as the last character, the string IS a palindrome if what's left when you throw away the first and last characters is a palindrome

yay - remove first character; remove last character; look at what's left

- a - IS a palindrome

Pseudocode

x=yay IS a palindrome

First character equals last character

- Create the substring by throwing away the first and last character
  - x[1:-1] or (x[1:len(x)-1])
- A

y = tt

Throw away first and last character - we have nothing left - we have an empty string left

battab - atta - tt - empty string - IS a palindrome

Another example - sum a list of numbers

What are the base cases?

What's the recursive case?

1, 2,3 4 5 = 1 + sum 2, 3, 4, 5 = 1+2+sum 3, 4, 5

1 + 2 + 3 + sum 4, 5 1 + 2 + 3 + 4 +5 + sum empty list = 0

If we have time:

More on importing Python libraries - note, this will NOT be on Exam 2 next week

Available Code

One of the big advantages of using Python is that there is a ton of code that exists “in the wild” that is available for your use.

- Math library - math — Mathematical functions — Python 3.10.0 documentation - math functions
- Numpy - https://numpy.org/ - arrays; linear algebra; FFT; random numbers; …
- Pandas - https://pandas.pydata.org/ - data analysis/data science
- Pillow - https://pypi.org/project/Pillow/ - image manipulation
- Matplotlib - https://matplotlib.org/index.html - graphics and plots
- Ggplot - http://ggplot.yhathq.com/ - more plots
- CSV files - https://docs.python.org/3/library/csv.html

See https://www.ubuntupit.com/best-python-libraries-and-packages-for-beginners/ for more

Importing Code

You can import any of these existing libraries into your Python program

You can also import any code you've previously written yourself (or that your professor or classmate has written)

If the package has already been installed, just use the import statement:

```
import math  #imports the existing math library; you can now use any of the
             #functions in that library
import hailstone #imports the hailstone.py program from last week and lets you
                 #use the "flight" function
```

There are multiple formats

Basic:

```
import math
```

Means you must include the module name in each function call:

```
x = math.sin(0)  # if you just said “sin” the Python interpreter wouldn’t know what you meant
```

Rename:

```
import math as m
```

Now you can call the functions with a simpler name: x = m.sin(o)

Import formats

Only import the functions you need:

```
from math import sin
```

Then you don’t have to refer to the module - x = sin(0) # valid

Or import all functions with the * wildcard character:

```
from math import *
```

Now you can refer to all the math functions without referring to the module

Just make sure you don’t have another function with the same name!!!!

An example using the pandas module

You can read in and process a .csv file with a single statement:

First, install pandas

Then in your program:

```
import pandas as pd
```

Now read in the file:

```
df =  pd.read_csv(“Spring_2022.csv”)
```

And we can easily drop those pain-in-the-neck rows of commas:

```
df.dropna()
```

Installing packages you don’t already have

Pip - package installer for Python

Conda - short for “anaconda”

- Lets you install python packages so that your Python interpreter knows where they are and can use them

Importing your own code

We'll write a program called "dates.py" that includes a function called "convert_date."

Then we'll write a separate program that imports "dates" and calls that function.

The separate program is:

```
import dates
if __name__ == "__main__":
        print(dates.convert_date('07042020'))
```

```
from dates import *
if __name__ == "__main__":
    print(convert_date('07042020'))
```

That's it - really!!!

‘The rules’ for importing your own code:

1. The name of the module MUST end in ‘.py’
2. The module must be in the Python path so that it can be found
    a. Be in the same directory as the calling program
    b. Be in a directory that Python always searches for programs
    c. Include the entire path to the file in the import statement

Fixing the example from class

```
import new_p2_demo

print("success")

db =
new_p2_demo.read_a_file("/Users/alfredarsenault/Documents/Fa
ll_2021_Project_2/Fall_2023.csv")

print(db)
```

Note: the import statement uses dots (.) not slashes to separate directories if the file you're searching for is not in the same directory as your program