Introduction to Computation for the Humanities and Social Sciences

CS 3
Chris Tanner
Lecture 23

Methods Man*

*the secret, 10th member of Wu-Tang
ANNOUNCEMENTS

- Projects **must** be submitted by Dec 3 at 11:59pm

- completed in-class assignment should be shown to a TA in order to get full HW credit (i.e., HW10)

- To have your project graded, sign up for a session w/ your Mentor TA (Dec 3 - Dec 10)

- Sign up for a presentation slot (Dec 4, 6, 11)
GOALS FOR TODAY

- Learn difference b/w functions and methods
- Understand why the choices in data structures and algorithms are super important
Lecture 23

- Functions vs Methods
- Data Structures
- Algorithms
Functions vs Methods

Functions

• We’ve exhaustively talked about functions, which are defined by their:
  • names
  • inputs
  • specific computations
  • outputs
Functions vs Methods

**Functions**

```python
1  def plus(a, b):
2     a = 2 * a
3     b = 3 + b
4     c = a + b
5     return c
6
7  def main():
8     a = 3
9     b = 5
10    c = plus(a, b)
11    c += 1
12    print(a)
13    print(b)
14    print(c)
15
16   if __name__ == "__main__":
17       main()
```
Functions vs Methods

Methods

text = “Code didn't work, no idea why…”
pattern = ‘a’
re.findall(pattern, text)

• Remember when we’d see things like `findall()` and I’d commonly call it a function then say “well, technically it’s not a function, but close enough”?

• Technically, `findall()` belongs to a specific object (a concrete instance of a Python file), i.e., `re`. so although it is a function, it is more aptly and specifically a method.
Objects

- Object-oriented programming is the most common paradigm of programming, and Python supports such.
- Covering this topic would require at least 1-2 lectures, but basically, any single .py file we’ve written can be nicely grouped into a single class

```python
1  class BasicMathExamples:
2    def plus(a, b):
3        return a + b
4  
5    def multiply(a, b):
6        return a * b
7  
8    def mystery_function(a, b):
9        return -1*a * 3*b
```
Functions vs Methods

Objects

• Anytime someone (including yourself) wishes to use this code, they can create a particular instance of your class, similar to when we import a package:

```python
test = BasicMathExamples()
```

• And we could access the functions (technically methods) via:

```python
test.plus(31, 97)
```

• `plus()` corresponds to that particular `test` object.
Functions vs Methods

Moral of the Story

• if you’re calling a function defined within your specific .py file, it’s referred to as a function

• if you’re calling a function specific to a particular object, e.g.

```
re.findall(pattern, text)

line = line.strip().split("\n")
```

they are technically methods
Lecture 23

- Functions vs Methods
- Data Structures
- Algorithms
Data Structures

- Remember: data structures are the constructs which house our data, e.g.,
  - single-valued (a single String, Int, Bool, Float)
  - lists
  - sets
  - dictionaries
  - tuples — we didn’t really cover these
Data Structures

• They’re immensely important because they allow us to store and retrieve our data in which ways are most intuitive to us, and hopefully efficient for the computation we’re working on.

• Some make more sense than others, for a given scenario, e.g., list vs dictionary for storing all student names.

• Some are faster than others, depending on what we’re trying to do, e.g., list vs set
• **Binary Trees** allow us to quickly sort items and keep them sorted whenever we add new items.
Lecture 23

- Functions vs Methods
- Data Structures
- Algorithms
• When we decide an approach to compute something, that is our algorithm.

• As mentioned, there are essentially an infinite number of ways to compute a given task.

• Some are more efficient approaches than others.
Example: write a program which determines if any two items in a list sum to a particular value

```python
special_num = 40
ages = [22, 18, 24, 34, 19, 21]
for i in ages:
    for j in ages:
        if i + j == special_num:
            print("Yes, two do!")
```

This requires going through the entire list... for every single item in the list!

So, if our list is of length \( N \), that’s \( N^2 \) operations/checks.

That’s not too efficient. Imagine if \( N = 1,000 \) items, that’s 1 million operations.
• Instead, if we stored our numbers in a `set()`, we could instantly check if a number exists within it.

```python
special_num = 40
ages = set({22, 18, 24, 34, 19, 21})
for i in ages:
    if special_num - i in ages:
        print("Yes, two do!")
```

• This requires just going through the list once

• So, if our list is of length $N$, that’s $N$ operations/checks.

• That’s very efficient. Imagine if $N = 1,000$ items, that’s only 1,000 operations, which is 1,000 times faster than the previous solution.
Algorithms

Main Takeaway

- The way you design your computations can be insanely important for speed-purposes.
- Some solutions (algorithms) can be so painfully slow, your program will never finish.
- In fact, some types of problems are so complex, there are no known solutions which can ever finish, e.g.,

Travelling Salesman Problem:

"Given a list of cities and the distances between each pair of cities, what is the shortest possible route that visits each city and returns to the origin city?"
LAB TIME