**Incremental Development**

Kenneth Thomas Barrett

Computer Science Department, University of the People

CS1101 - Programming Fundamentals

Ola Fatai Ayanleke Ayanleke

December 12, 2024

### Part 1

# Step 1: create a basic skeleton for our function. Function is defined with

# two parameters 'a' and 'b', representing the lengths of the two legs

# of the right triangle. We will go ahead and return 0 to ensure the function

# is returning a defined value as anticipated.

# def calc\_hypotenuse(a, b):

# return 0

# print(calc\_hypotenuse(1,2)) # Test input; Output: 0

# Step 2: implement the hypotenuse formula (c=sqrt(a^2+b^2)); import numpy

# to use the sqrt function. Use print statements to ensure calculations are as expected at each stage. Our return statement

# can be assumed to represent c due to the sole purpose of the function being calculating the

# hypotenuse. Justification for using NumPy over the math library: numpy was chosen due to it generally being a bit faster than math;

# while these calculations are simple enough that it doesn't make a real difference, it is still a good habit. Not

# to mention, many other popular Python libraries (such as pandas) need numpy to be used as opposed to math.

# import numpy as np

# def calc\_hypotenuse(a, b):

# # Expected Output: a^2 = 9; b^2 = 16

# print(f'a^2 = {a\*\*2}; b^2 = {b\*\*2}') # Actual Output (Using input shown below): a^2 = 9; b^2 = 16

# # Expected Output (Using input shown below): a^2 + b^2 = 25

# print(f'a^2 + b^2 = {a\*\*2 + b\*\*2}') # Actual Output (Using input shown below): a^2 + b^2 = 25

# return np.sqrt(a\*\*2 + b\*\*2)

# # Test input; expected output: 5

# print(calc\_hypotenuse(3, 4)) # Output: 5.0

# Step 3: Now that we know each stage of our calculations are working as anticipated, we can remove the print statements.

# We will then test the function with known values, and compare the actual output with the output we are expecting.

# import numpy as np

# def calc\_hypotenuse(a, b):

# return np.sqrt(a\*\*2 + b\*\*2)

# print(calc\_hypotenuse(3, 4)) # Expected Output: c = 5; Actual Output: 5.0

# print(calc\_hypotenuse(3, 8)) # Expected Output: c ≈ 8.544; Actual Output: 8.54400374531753

# print(calc\_hypotenuse(20, 40)) # Expected Output: c ≈ 44.72; Actual Output: 44.721359549995796

# Step 4: Create docstrings for the function; while normally docstrings would be created earlier in this process to document

# for collaboration with other developers, an argument can be made that this function is simple enough to justify

# creating docstrings at any point before pushing changes to a shared development or production build.

import numpy as np

def calc\_hypotenuse(a, b):

"""

Calculate the length of the hypotenuse of a right triangle.

Given the lengths of the two legs (a and b) of a right triangle,

this function computes the length of the hypotenuse using the

Pythagorean theorem: c = sqrt(a^2 + b^2), where:

- a: Length of one leg of the triangle.

- b: Length of the other leg of the triangle.

Parameters:

a (float or int): Length of the first leg of the triangle.

b (float or int): Length of the second leg of the triangle.

Returns:

float: The length of the hypotenuse.

Example:

>>> calc\_hypotenuse(3, 4)

5.0

Note:

The function assumes that the input values are non-negative.

"""

return np.sqrt(a\*\*2 + b\*\*2)

print(calc\_hypotenuse(3, 4)) # Expected Output: c = 5; Actual Output: 5.0

print(calc\_hypotenuse(3, 8)) # Expected Output: c ≈ 8.544; Actual Output: 8.54400374531753

print(calc\_hypotenuse(20, 40)) # Expected Output: c ≈ 44.72; Actual Output: 44.721359549995796

### Part 2

# This function is intended for use by Search Engine Optimization specialists. The meta description of a webpage is recommended

# to remain at or under 155 characters; this function will evaluate whether a provided URL's meta description follows these standards, and

# informs the SEO specialist whether the length needs to be shortened.

# Step 1: Create a basic skeleton for our function; this function will be defined with

# a string representing a URL as a parameter. Returning None just to ensure the return is working as expected.

# def meta\_desc\_len(url):

# return None

# Test Input (Just using the World of Warcraft official website as a test URL):

# print(meta\_desc\_check('https://worldofwarcraft.blizzard.com/en-us/')) # Output: None (as expected)

# Step 2: import requests; fetch the page content of a url using a GET request. Return the response (this will not be the contents themselves, only the response with response code).

# import requests

# def meta\_desc\_len(url):

# response = requests.get(url)

# return response

# Test Input:

# print(meta\_desc\_len('https://worldofwarcraft.blizzard.com/en-us/')) # Output: <Response [200]> (as expected)

# Step 3: if the response code is 200 as expected, import BeautifulSoup (will require

# installation; can be installed with pip using `pip install beautifulsoup4`) and parse the HTML content.

# If the response code is not 200, we will not be able to parse the HTML, so create a condition that

# handles other status codes (it's very unlikely this will be needed but it's still important to handle).

# The output should be all HTML from the webpage provided.

# import requests

# from bs4 import BeautifulSoup

# def meta\_desc\_len(url):

# response = requests.get(url)

# if response.status\_code == 200:

# soup = BeautifulSoup(response.text, 'html.parser') # Parse HTML

# return soup

# else:

# return f"Failed to retrieve the webpage. Status code: {response.status\_code}"

# # Test Input (Just using the World of Warcraft official website as a test URL):

# print(meta\_desc\_len('https://worldofwarcraft.blizzard.com/en-us/')) Output (as expected): all HTML code from the webpage -- too large to include here. Full output: https://pastebin.com/XjZ09y5L

# Step 4: Using soup.find(), we can query the HTML to search for specific elements. Because meta descriptions are always stored as <meta name="description">, we can easily

# find the meta description tag of a page if it exists. The output should be the relevant HTML from our meta description query.

# import requests

# from bs4 import BeautifulSoup

# def meta\_desc\_len(url):

# response = requests.get(url)

# if response.status\_code == 200:

# soup = BeautifulSoup(response.text, 'html.parser') # Parse HTML

# meta\_description = soup.find('meta', attrs={'name': 'description'}) # Query contents to find meta description

# return meta\_description

# else:

# return f"Failed to retrieve the webpage. Status code: {response.status\_code}"

# # Test Input (Just using the World of Warcraft official website as a test URL):

# print(meta\_desc\_len('https://worldofwarcraft.blizzard.com/en-us/')) # Output (as expected): matching HTML code relative to the query; too large to include here. Full output: https://pastebin.com/TBxySmUJ

# Step 5: If the meta description exists, return its content. If not, inform the user no meta description was found. The output should be the text contained in the

# meta description.

# import requests

# from bs4 import BeautifulSoup

# def meta\_desc\_len(url):

# response = requests.get(url)

# if response.status\_code == 200:

# soup = BeautifulSoup(response.text, 'html.parser') # Parse HTML

# meta\_description = soup.find('meta', attrs={'name': 'description'}) # Query contents to find meta description

# if meta\_description:

# # If the description meta tag is found...

# # Return the contents

# return meta\_description.get('content')

# else:

# return "Meta description not found."

# else:

# return f"Failed to retrieve the webpage. Status code: {response.status\_code}"

# # Test Input (Just using the World of Warcraft official website as a test URL):

# print(meta\_desc\_len('https://worldofwarcraft.blizzard.com/en-us/')) # Output: "Join thousands of mighty heroes in Azeroth, a world of magic and limitless adventure." (as expected)

# Step 6: As mentioned, SEO best-practices state that meta descriptions should be under approximately 155 characters. Using the meta description content attained during step 6,

# calculate the number of characters contained. We will also be storing the meta description to be a part of our return statement later, as opposed to it being the return statement itself.

# The output should be the number of characters in the meta description text.

# import requests

# from bs4 import BeautifulSoup

# def meta\_desc\_len(url):

# response = requests.get(url)

# if response.status\_code == 200:

# soup = BeautifulSoup(response.text, 'html.parser') # Parse HTML

# meta\_description = soup.find('meta', attrs={'name': 'description'}) # Query contents to find meta description

# if meta\_description:

# # If the description meta tag is found...

# contents = meta\_description.get('content') # Store the contents from meta description query

# # Return the number of characters

# return len(contents)

# else:

# return "Meta description not found."

# else:

# return f"Failed to retrieve the webpage. Status code: {response.status\_code}"

# # Test Input (Just using the World of Warcraft official website as a test URL):

# print(meta\_desc\_len('https://worldofwarcraft.blizzard.com/en-us/')) # Output: 85 (as expected)

# Step 7: Evaluate whether the number of characters contained in the meta description fetched from the webpage is at or under 155 characters. Inform the user either way in our return.

# Additionally, similar to our handling of the text itself in step 6, we will be storing the character length of the meta description as well, as opposed to

# simply returning it. This will be used as part of our return statement later.

# import requests

# from bs4 import BeautifulSoup

# def meta\_desc\_len(url):

# response = requests.get(url)

# if response.status\_code == 200:

# soup = BeautifulSoup(response.text, 'html.parser') # Parse HTML

# meta\_description = soup.find('meta', attrs={'name': 'description'}) # Query contents to find meta description

# if meta\_description:

# # If the description meta tag is found...

# contents = meta\_description.get('content') # Store the contents from meta description query

# # Return the number of characters

# contents\_len = len(contents) # Store the meta description length

# if contents\_len <= 155:

# return 'Meta description length is under 155 characters and thus within recommendations for SEO best-practices.'

# else:

# 'Meta description length is too long. Keep the number of characters at or under 155 to avoid truncation.'

# else:

# return "Meta description not found."

# else:

# return f"Failed to retrieve the webpage. Status code: {response.status\_code}"

# # Test Input (Just using the World of Warcraft official website as a test URL):

# print(meta\_desc\_len('https://worldofwarcraft.blizzard.com/en-us/')) # Output: Meta description length is within recommendations for SEO best-practices. (as expected)

# Step 8: format the return statement to be helpful to the user. We will be using the contents of the meta description, the length, and our evaluation of whether

# SEO best-practices for meta description length have been met, and we will return these to the user in a well-formatted, actionable way.

# import requests

# from bs4 import BeautifulSoup

# def meta\_desc\_len(url):

# response = requests.get(url)

# if response.status\_code == 200:

# soup = BeautifulSoup(response.text, 'html.parser') # Parse HTML

# meta\_description = soup.find('meta', attrs={'name': 'description'}) # Query contents to find meta description

# if meta\_description:

# # If the description meta tag is found...

# contents = meta\_description.get('content') # Store the contents from meta description query

# # Return the number of characters

# contents\_len = len(contents) # Store the meta description length

# if contents\_len <= 155:

# return f'"{contents}" is {contents\_len} characters long. Meta description length is under 155 characters and thus within recommendations for SEO best-practices.'

# else:

# f'"{contents}" is {contents\_len} characters long. Meta description length is too long. Keep the number of characters at or under 155 to avoid truncation.'

# else:

# return "Meta description not found."

# else:

# return f"Failed to retrieve the webpage. Status code: {response.status\_code}"

# # Test Input (Just using the World of Warcraft official website as a test URL):

# print(meta\_desc\_len('https://worldofwarcraft.blizzard.com/en-us/')) # Output: "Join thousands of mighty heroes in Azeroth, a world of magic and limitless adventure." is 85 characters long. Meta description length is under 155 characters and thus within recommendations for SEO best-practices.

# Step 9: test the function using several URLs. The output should once again include the contents of the meta description and the number of characters in it, in addition

# to direct feedback about whether the length is too long.

# import requests

# from bs4 import BeautifulSoup

# def meta\_desc\_len(url):

# response = requests.get(url)

# if response.status\_code == 200:

# soup = BeautifulSoup(response.text, 'html.parser') # Parse HTML

# meta\_description = soup.find('meta', attrs={'name': 'description'}) # Query contents to find meta description

# if meta\_description:

# # If the description meta tag is found...

# contents = meta\_description.get('content') # Store the contents from meta description query

# # Return the number of characters

# contents\_len = len(contents) # Store the meta description length

# if contents\_len <= 155:

# return f'"{contents}" is {contents\_len} characters long. Meta description length is under 155 characters and thus within recommendations for SEO best-practices.'

# else:

# f'"{contents}" is {contents\_len} characters long. Meta description length is too long. Keep the number of characters at or under 155 to avoid truncation.'

# else:

# return "Meta description not found."

# else:

# return f"Failed to retrieve the webpage. Status code: {response.status\_code}"

# # Test Input:

# print(meta\_desc\_len('https://worldofwarcraft.blizzard.com/en-us/')) # Output as expected: "Join thousands of mighty heroes in Azeroth, a world of magic and limitless adventure." is 85 characters long. Meta description length is under 155 characters and thus within recommendations for SEO best-practices.

# print(meta\_desc\_len('https://uopeople.edu/')) # Output as expected: "University of the People is a tuition-free online university for students from all over the world. Learn more about University of the People!" is 141 characters long. Meta description length is under 155 characters and thus within recommendations for SEO best-practices.

# print(meta\_desc\_len('https://www.tensorflow.org/')) # "An end-to-end open source machine learning platform for everyone. Discover TensorFlow's flexible ecosystem of tools, libraries and community resources." is 151 characters long. Meta description length is under 155 characters and thus within recommendations for SEO best-practices.

# Step 10: Create docstrings to provide documenation on what the function does and how to use it.

import requests

from bs4 import BeautifulSoup

def meta\_desc\_len(url):

"""

Fetches the meta description of a webpage and returns its length along with a message

about whether the length adheres to SEO best practices.

This function makes a GET request to the provided URL, parses the HTML content,

and looks for a <meta> tag with the name attribute set to "description". If found, it

returns the content of the meta description, its length, and an evaluation of whether

the length is under the recommended 155-character limit for SEO. If no meta description

is found or if the webpage cannot be accessed, appropriate error messages are returned.

Args:

url (str): The URL of the webpage to analyze.

Returns:

str: A message with the meta description content, its length, and SEO assessment

or an error message if the meta description is not found or the request fails.

Example:

>>> meta\_desc\_len('https://worldofwarcraft.blizzard.com/en-us/')

'"Join thousands of mighty heroes in Azeroth, a world of magic and limitless adventure." is 85 characters long. Meta description length is under 155 characters and thus within recommendations for SEO best-practices.'

"""

response = requests.get(url)

if response.status\_code == 200:

soup = BeautifulSoup(response.text, 'html.parser') # Parse HTML

meta\_description = soup.find('meta', attrs={'name': 'description'}) # Query contents to find meta description

if meta\_description:

# If the description meta tag is found...

contents = meta\_description.get('content') # Store the contents from meta description query

# Return the number of characters

contents\_len = len(contents) # Store the meta description length

if contents\_len <= 155:

return f'"{contents}" is {contents\_len} characters long. Meta description length is under 155 characters and thus within recommendations for SEO best-practices.'

else:

return f'"{contents}" is {contents\_len} characters long. Meta description length is too long. Keep the number of characters at or under 155 to avoid truncation.'

else:

return "Meta description not found."

else:

return f"Failed to retrieve the webpage. Status code: {response.status\_code}"

# Test Input:

print(meta\_desc\_len('https://worldofwarcraft.blizzard.com/en-us/')) # Output as expected: "Join thousands of mighty heroes in Azeroth, a world of magic and limitless adventure." is 85 characters long. Meta description length is under 155 characters and thus within recommendations for SEO best-practices.

print(meta\_desc\_len('https://uopeople.edu/')) # Output as expected: "University of the People is a tuition-free online university for students from all over the world. Learn more about University of the People!" is 141 characters long. Meta description length is under 155 characters and thus within recommendations for SEO best-practices.

print(meta\_desc\_len('https://www.tensorflow.org/')) # "An end-to-end open source machine learning platform for everyone. Discover TensorFlow's flexible ecosystem of tools, libraries and community resources." is 151 characters long. Meta description length is under 155 characters and thus within recommendations for SEO best-practices.

[GitHub Repo](https://github.com/KennethTBarrett/CS1101/tree/main/Unit_4/Assignment) since it’s very difficult to read here, especially seeing as part 2 has 218 lines of code. Includes documentation of the full output for part 2 steps 3 and 4 (it’s over 1100 lines, so I figured you’d rather not have it with the 218 lines of my actual code).