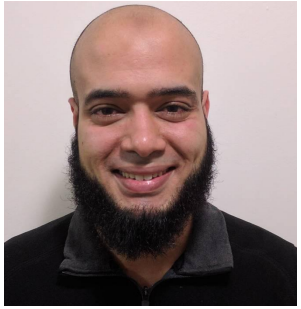
Python Programming

Recursive Functions

Homework 3

Mostafa S. Ibrahim
Teaching, Training and Coaching since more than a decade!

Artificial Intelligence & Computer Vision Researcher
PhD from Simon Fraser University - Canada
Bachelor / Msc from Cairo University - Egypt
Ex-(Software Engineer / ICPC World Finalist)



Problem #1: Count primes

- Implement function: **def** count_primes(start, end)
  - It counts prime numbers in this range
- Don’t use loops at all.
- Don’t use any python functions

```
print(count_primes(10, 20))          # 4
print(count_primes(10, 200))         # 42
```

Problem #2: Greedy Robot

- Read an integer matrix (all **distinct** values)
- A robot starts at cell (0, 0).
- Take the value in the current cell and moves.
  - It can move only **one step** to either: *Right, Bottom or the diagonal.*
  - It always selects the destination cell that has **maximum value**.
- Print the total values the robot collects

3
1 2 3
4 **5** 6
7 8 **9**
⇒ (0, 0) (1, 1), (2, 2) ⇒ 15

3
1 2 3
5 4 9
7 6 8
⇒ (0,0)⇒(1,0)⇒(2,0)⇒(2,1)⇒(2,2)
⇒27

2
1 2 3 4 5
6 7 8 9 10
⇒ 35

Problem #2: Greedy Robot

- Write a function that takes a matrix and compute the path sum

```
rows, cols, matrix = read_matrix()
print(get_path_sum(matrix))
```

Problem #3: Standard Max

- In this task, we would like to implement a max function to almost behave like standard one: what we pass, return and raised errors!
  - The recursive part of this function is trivial, like what we met
  - Make use of this task to think how things in professional development are done

builtins.py

```
def max(*args, key=None): # known special case of max
    """
    max(iterable, *[, default=obj, key=func]) -> value
    max(arg1, arg2, *args, *[, key=func]) -> value

    With a single iterable argument, return its biggest item. The
    default keyword-only argument specifies an object to return if
    the provided iterable is empty.
    With two or more arguments, return the largest argument.
    """
```

Problem #3: Standard Max

```
#my_max = max   # uncomment to test python max

print(my_max(2, 5))                          # 5
print(my_max([10, 3, 60, 20]))               # 60
print(my_max(10, 3, 6, 20))                  # 20
print(my_max({5, 7, 1}))                     # 7
print(my_max([5, 1], [4, 9]))                # [5, 1]
print(my_max('1234'))                        # 4
print(my_max('1234', '98'))                  # 98
print(my_max('1234', '98', key = len))       # 1234
print(my_max([5, 1], [4, 9], key = sum))     # [4, 9]

# Don't show any other internal exceptions
#print(my_max())                  # TypeError: max expected 1 argument, got 0
#print(my_max(default = -1))      # TypeError: max expected 1 argument, got 0
#print(my_max([]))                # ValueError: max() arg is an empty sequence
print(my_max([], default = None)) # None
#print(my_max(-15))     # TypeError: 'int' object is not iterable
#print(my_max(3, [4])) # TypeError: '>' not supported between instances of 'list' and 'int'
```

Problem #4: Deep Reverse v1

- The standard reverse function/method only reverse the top level
- What if we have list of list list and we would like to reverse all of them, regardless how deep?
- Develop in-place function

```
lst = [1, [2, 3, 4], [5, 6]]
lst.reverse()   # top level reverse ONLY
print(lst)  # [[5, 6], [2, 3, 4], 1]

lst = [1, [2, 3, 4], [5, 6]]
deep_reverse(lst)   # reverse very deep lists
print(lst)  # [[6, 5], [4, 3, 2], 1]

lst = [1, [2, 3, 4], [5, [6, 7, 8]]]
deep_reverse(lst)
print(lst)  # [[[8, 7, 6], 5], [4, 3, 2], 1]

lst = [1, [2, 3, 4], [5, [6, 7, [8, 9.5, 'hey']]]]
deep_reverse(lst)
print(lst)  # [[[['hey', 9.5, 8], 7, 6], 5], [4, 3, 2], 1]
```

Problem #5: Deep Reverse v2

- The exact problem, but consider:
  - The function is not inplace. Return a new deeply reversed list
  - Implement in a single line!

```
lst = [1, [2, 3, 4], [5, 6]]
print(deep_reverse(lst))    # [[6, 5], [4, 3, 2], 1]

lst = [1, [2, 3, 4], [5, [6, 7, 8]]]
print(deep_reverse(lst))    # [[[8, 7, 6], 5], [4, 3, 2], 1]

lst = [1, [2, 3, 4], [5, [6, 7, [8, 9.5, 'hey']]]]
print(deep_reverse(lst))    # [[[['hey', 9.5, 8], 7, 6], 5], [4, 3, 2], 1]
```

Problem #6: Fibonacci

- Implement fibonacci: def fibonacci(n)
  - Recall fibonacci sequence: 1 1 2 3 **5 8 13** 21 35
  - E.g. fibonacci(6) = 13
  - Recall that: fibonacci(n) = fibonacci(n-1) + fibonacci(n-2). E.g. fib(6) = fib(5)+fib(4) =13
    - So it calls 2 subproblems of its type
- Can u compute fibonacci(35)? fibonacci(40)? fibonacci(50)? More?
  - Why? Any work around? Hint: Save the intermediate results

"Acquire knowledge and impart it to the people."

"Seek knowledge from the Cradle to the Grave."