Sharif University of Technology
Department of Computer Engineering

Fundamentals of Programming

Python Language

Arman Malekzadeh
PhD Candidate in Artificial Intelligence

Table of contents

Combination of Loops and Conditional Logic (Cont.)

Exercise 9: Arbitrary Base Conversion from 2 to 16

- Write a function that takes a string representing a number in base 2 through 16 and converts it to its equivalent in base 10.
- The function should be able to handle bases up to 16.
- The function should return -1 if the string is not a valid number in the given base.

Solution to Exercise 9: Arbitrary Base Conversion from 2 to 16

```
def base_conversion(s, b):
    """Return the base 10 representation of s in base b.
    """
    result = 0
    for i, digit in enumerate(s[::-1]):
        if digit.isdigit():
            value = int(digit)
        else:
            value = ord(digit) - ord('A') + 10
        if value >= b:
            return -1
        result += value * b ** i
    return result
```

Solution to Exercise 9: Arbitrary Base Conversion from 2 to 16

Tracing the code for `base_conversion('101', 2)` gives:

```
i = 0, digit = '1', value = 1, result = 1
i = 1, digit = '0', value = 0, result = 1
i = 2, digit = '1', value = 1, result = 5
```

Exercise 10: Reduce a Fraction to Lowest Terms

- Write a function that takes a fraction as a string and returns the fraction reduced to lowest terms.

- The function should return -1 if the string is not a valid fraction.

Solution to Exercise 10: Reduce a Fraction to Lowest Terms

```
def gcd(a, b):
    """Return the greatest common divisor of a and b.
    """
    while b != 0:
        a, b = b, a % b
    return a

def reduce_fraction(fraction):
    """Return the fraction reduced to lowest terms.
    """
    numerator, denominator = fraction.split('/')
    numerator, denominator = int(numerator), int(denominator)
    divisor = gcd(numerator, denominator)
    return str(numerator // divisor) + '/' + str(denominator // divisor)
```

Exercise 11: Sieve of Eratosthenes

- The sieve of Eratosthenes is one of the most efficient ways to find all primes smaller than n when n is smaller than 10 million or so.
- The idea is to start with a list of all numbers up to n and repeatedly remove multiples of primes from the list.
- The running time of this algorithm is $\mathcal{O}(n \log \log n)$.

Code for Exercise 11: Sieve of Eratosthenes

```
def sieve(n):
    """Return a list of all primes less than n.
    """
    primes = [True] * n
    primes[0] = primes[1] = False
    for i in range(2, n):
        if primes[i]:
            for j in range(i*i, n, i):
                primes[j] = False
    return [i for i, prime in enumerate(primes) if prime]
```

Useful Stuff

Zip in Python

- The `zip()` function takes iterables (can be zero or more), aggregates them in a tuple, and return it.
- The `zip()` function returns an iterator of tuples based on the iterable objects.
- If we do not pass any parameter, zip() returns an empty iterator.

Zip in Python

```
# Python code to demonstrate the working of
# zip()

# initializing lists
name = [ "Manjeet", "Nikhil", "Shambhavi", "Astha" ]
roll_no = [ 4, 1, 3, 2 ]

# using zip() to map values
mapped = zip(name, roll_no)

# converting values to print as set
mapped = set(mapped)

# printing resultant values
print ("The zipped result is : ",end="")
print (mapped) # prints {('Manjeet', 4), ('Nikhil', 1), ('Astha', 2), ('
                                        Shambhavi', 3)}
```

Zip in Python

```
# unzipping values
namez, roll_noz = zip(*mapped)

print ("The unzipped result: \n",end="")

# printing initial lists
print ("The name list is : ",end="")
print (namez) # prints ('Manjeet', 'Nikhil', 'Shambhavi', 'Astha')

print ("The roll_no list is : ",end="")
print (roll_noz) # prints (4, 1, 3, 2)
```

Dynamic Programming: Fibonacci

- Dynamic programming is a method for solving a complex problem by breaking it down into a collection of simpler subproblems, solving each of those subproblems just once, and storing their solutions.

- The next time the same subproblem occurs, instead of recomputing its solution, one simply looks up the previously computed solution, thereby saving computation time.

Previous Example: Fibonacci

```
def fib(n):
    """Return the nth Fibonacci number.
    """
    if n == 0:
        return 0
    elif n == 1:
        return 1
    else:
        return fib(n-1) + fib(n-2)
```

Dynamic Programming: Fibonacci

```
def fib_memo(n, memo):
    """Return the nth Fibonacci number.
    """
    if n in memo:
        return memo[n]
    elif n == 0:
        result = 0
    elif n == 1:
        result = 1
    else:
        result = fib_memo(n-1, memo) + fib_memo(n-2, memo)
    memo[n] = result
    return result
```

Time Magic in Jupyter Notebook

```
%timeit fib(20)
```

```
%timeit fib_memo(20, {})
```

%%time Magic in Jupyter Notebook

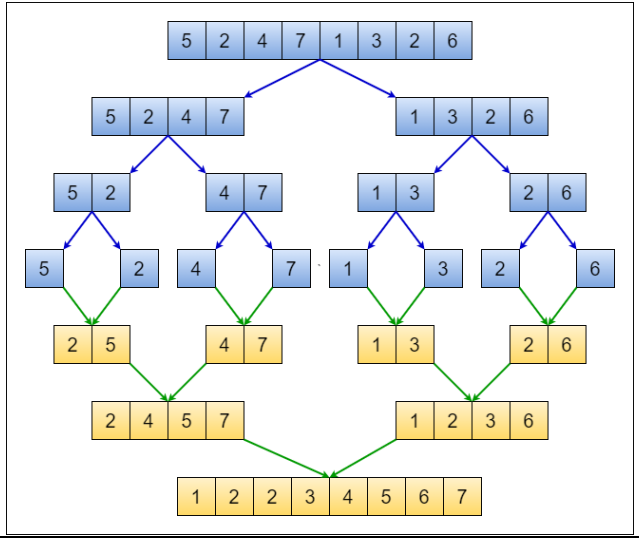
```
%%time
fib(20)
```

```
%%time
fib_memo(20, {})
```

Merge Sort

- Merge sort is a sorting technique based on divide and conquer technique.
- With worst-case time complexity being $\mathcal{O}(n \log n)$, it is one of the most respected algorithms.
- Merge sort first divides the array into equal halves and then combines them in a sorted manner.

Merge Sort


5 2 4 7 1 3 2 6
5 2 4 7
1 3 2 6
5 2
4 7
1 3
2 6
5
2
4
7
1
3
2
6
2 5
4 7
1 3
2 6
2 4 5 7
1 2 3 6
1 2 2 3 4 5 6 7

Merge Sort

```
def merge_sort(arr):
    """Return a sorted copy of arr.
    """
    if len(arr) <= 1:
        return arr
    mid = len(arr) // 2
    left = merge_sort(arr[:mid])
    right = merge_sort(arr[mid:])
    return merge(left, right)
```

Merge Sort

```
    def merge(left, right):
        """Return a sorted list of left and right.
        """
        result = []
        i, j = 0, 0
        while i < len(left) and j < len(right):
            if left[i] <= right[j]:
                result.append(left[i])
                i += 1
            else:
                result.append(right[j])
                j += 1
        result += left[i:]
        result += right[j:]
        return result
```

Using the Random Shuffle in Python

- The `random` module provides access to functions that support many operations.
- Perhaps the most important thing is that it allows you to generate random numbers.
- The `random` module contains a function called `shuffle` that takes a list and re-arranges the order of the elements.

```
import random
arr = [1, 2, 3, 4, 5]
random.shuffle(arr)
print(arr) # prints [3, 2, 5, 1, 4]
```

References

References I

[1] B Downey, A. (2015). Think Python: How to Think Like a Computer Scientist-2nd Edition.

[2] Deitel, H. M., & Deitel, P. J. (2004). C: How to program. Pearson Educacion.

Sharif University of Technology
Department of Computer Engineering

Arman Malekzadeh

Fundamentals of Programming

Python Language