15-110: Principles of Computing

Lecture 4: Abstraction (II)
From Algorithms to Python

August 09, 2022
Today...

• **Last session:**
  • Simplifications, Abstractions

• **Today’s session:**
  • Abstraction (Wrap up), Python!

• **Announcements:**
  • HW1 is due Today at 10 pm.
  • Quiz I grades are out
  • OH in the ARC Hallway
  • Course website: [https://web2.qatar.cmu.edu/~mhhammou/15110-f22/index.html](https://web2.qatar.cmu.edu/~mhhammou/15110-f22/index.html)
Find another problem that can be solved using the same algorithm. Be creative!

First you take your ____________ then add a layer of ____________ before you pour on a hearty dose of ____________.
Next, press some ____________ down into the ____________ before covering with a sprinkle of ____________.
That's how I make a ________________ !

And the winner is...

- Pasta, Chicken Marinara, Pizza, Burgers
- Salads
- Cake, cupcake, milkshake, pancakes, protein shake, smoothie, Ice-cream
- Coffee, Tea
- ... 
- CMU Student (college person)
- PC
- Morning call
- Homework
- ... 
- Wedding Ring 
- Paintings (abstract, realism)
Abstraction

• Let us consider again the problem of choosing snacks from a cafeteria given a certain budget and calorie intake

• The problem can be phrased as follows:
  • You want to buy the highest-calorie snack from the below and pay a max of 15 QAR

<table>
<thead>
<tr>
<th>Item</th>
<th>Price (QAR)</th>
<th>Calories</th>
</tr>
</thead>
<tbody>
<tr>
<td>Muffin</td>
<td>6</td>
<td>480</td>
</tr>
<tr>
<td>Croissant</td>
<td>7</td>
<td>595</td>
</tr>
<tr>
<td>Chips</td>
<td>10</td>
<td>950</td>
</tr>
<tr>
<td>Hamburger</td>
<td>8</td>
<td>800</td>
</tr>
<tr>
<td>Chocolate</td>
<td>2</td>
<td>300</td>
</tr>
<tr>
<td>Fruit salad</td>
<td>5</td>
<td>200</td>
</tr>
</tbody>
</table>

Objective: *Maximize* calories without exceeding a certain budget (*constraint*)

How would you solve this?
Abstraction

• **Hint**: think about the calories per riyal
  • Since it is a maximization problem, the higher the better
  • In particular, you want the highest number of calories per each riyal (a *greedy strategy*) in order to obtain the highest-calorie snack with the 15QAR

<table>
<thead>
<tr>
<th>Item</th>
<th>Price (QAR)</th>
<th>Calories</th>
<th>Calories/Riyal</th>
</tr>
</thead>
<tbody>
<tr>
<td>Muffin</td>
<td>6</td>
<td>480</td>
<td>480/6 = 80</td>
</tr>
<tr>
<td>Croissant</td>
<td>7</td>
<td>595</td>
<td>595/7 = 85</td>
</tr>
<tr>
<td>Chips</td>
<td>10</td>
<td>950</td>
<td>700/10 = 95</td>
</tr>
<tr>
<td>Hamburger</td>
<td>8</td>
<td>800</td>
<td>800/8 = 100</td>
</tr>
<tr>
<td>Chocolate</td>
<td>2</td>
<td>300</td>
<td>300/2 = 150</td>
</tr>
<tr>
<td>Fruit salad</td>
<td>5</td>
<td>200</td>
<td>200/5 = 40</td>
</tr>
</tbody>
</table>

**Item Price (QAR) Calories Calories/Riyal**

- Muffin: 6 QAR, 480 calories, 480/6 = 80 calories/riyal
- Croissant: 7 QAR, 595 calories, 595/7 = 85 calories/riyal
- Chips: 10 QAR, 950 calories, 700/10 = 95 calories/riyal
- Hamburger: 8 QAR, 800 calories, 800/8 = 100 calories/riyal
- Chocolate: 2 QAR, 300 calories, 300/2 = 150 calories/riyal
- Fruit salad: 5 QAR, 200 calories, 200/5 = 40 calories/riyal

**Choice:**
- 5/10 of Chips: 5 QAR, 950 Cal, 5/10 * 950 Cal = 5 QAR
- Hamburger: 8 QAR, 800 Cal, 8 QAR
- Chocolate: 2 QAR, 300 Cal, 2 QAR

**Total:**

- **Calories:** 1,575 Cal
- **Price:** 15 QAR
Abstraction

This worked only because we assumed we can take fractions of items.

If this is not the case, this greedy strategy will not work!

This problem is known as the 0–1 knapsack problem.

<table>
<thead>
<tr>
<th>Item</th>
<th>Price (QAR)</th>
<th>Calories</th>
<th>Calories/Riyal</th>
</tr>
</thead>
<tbody>
<tr>
<td>Muffin</td>
<td>6</td>
<td>480</td>
<td>480/6 = 80</td>
</tr>
<tr>
<td>Croissant</td>
<td>7</td>
<td>595</td>
<td>595/7 = 85</td>
</tr>
<tr>
<td>Chips</td>
<td>10</td>
<td>950</td>
<td>950/10 = 95</td>
</tr>
<tr>
<td>Hamburger</td>
<td>8</td>
<td>800</td>
<td>800/8 = 100</td>
</tr>
<tr>
<td>Chocolate</td>
<td>2</td>
<td>300</td>
<td>300/2 = 150</td>
</tr>
<tr>
<td>Fruit salad</td>
<td>5</td>
<td>200</td>
<td>200/5 = 40</td>
</tr>
</tbody>
</table>

200 Cal 5 QAR
+ 800 Cal 8 QAR
+ 300 Cal 2 QAR

1,300 Cal 15 QAR
Abstraction

• Here is another combination that has more calories, albeit spending the same amount of money (thus, the greedy approach did not give us the best answer)

• Solving this problem requires applying another algorithmic approach known as “dynamic programming”, which is beyond the scope of this class

<table>
<thead>
<tr>
<th>Item</th>
<th>Price (QAR)</th>
<th>Calories</th>
<th>Calories/Riyal</th>
</tr>
</thead>
<tbody>
<tr>
<td>Muffin</td>
<td>6</td>
<td>480</td>
<td>480/6 = 80</td>
</tr>
<tr>
<td>Croissant</td>
<td>7</td>
<td>595</td>
<td>595/7 = 85</td>
</tr>
<tr>
<td>Chips</td>
<td>10</td>
<td>950</td>
<td>950/10 = 95</td>
</tr>
<tr>
<td>Hamburger</td>
<td>8</td>
<td>800</td>
<td>800/8 = 100</td>
</tr>
<tr>
<td>Chocolate</td>
<td>2</td>
<td>300</td>
<td>300/2 = 150</td>
</tr>
<tr>
<td>Fruit salad</td>
<td>5</td>
<td>200</td>
<td>200/5 = 40</td>
</tr>
</tbody>
</table>

595 Cal + 800 Cal = 1395 Cal
7 QAR + 8 QAR = 15 QAR
Abstraction

• Let us consider another problem where we have a set of items with different weights and values

• Your job is to take the highest valuable load in a bag without exceeding a weight of 15Kg

<table>
<thead>
<tr>
<th>Item</th>
<th>Weight (Kg)</th>
<th>Value (QAR)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sceptre</td>
<td>4</td>
<td>10</td>
</tr>
<tr>
<td>Shoes</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Helmet</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>Armour</td>
<td>12</td>
<td>4</td>
</tr>
<tr>
<td>Dagger</td>
<td>2</td>
<td>2</td>
</tr>
</tbody>
</table>

Objective: Maximize value without exceeding a certain weight (constraint)

How would you solve this?
Abstraction

• Do you think that this problem is similar to the snack problem?
  • They are actually the same!

• If we can craft an algorithm for the snack problem, we can transform it (with minimal effort) into a solution for this problem

• The *core* of the two problems is:
  a. There is a set of items to choose from, with two associated values
  b. The answer consists of a subset of items such that one value is minimized/maximized and the other value adds up to a certain amount $k$
  c. The items cannot be split (i.e., non-fractional items)
Abstraction

• *Abstraction* is the ability to overlook the unimportant details of a problem and focus only on the important core parts of it

• By doing this, we can transform the problem into something else, which we have a solution for
Abstraction

• Assume you get as input a product’s expiry date (day, month, and year, all in numbers). Is the product expired?
  • Compare the expiry date and today’s date, determine which one comes first
  • Same as Quiz 1, oldest of two persons!
The Compounding Process

- Assume you want to deposit $100 in a bank that offers a 10% interest rate that is *compounded annually*
  - What would be your total amount of money after 3 years?

<table>
<thead>
<tr>
<th>Year</th>
<th>Your Money</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>$100</td>
</tr>
<tr>
<td>1</td>
<td>$100 + ($100×0.1) = $100 × (1+0.1) = $100 × 1.1 = $110</td>
</tr>
<tr>
<td>2</td>
<td>$110 × 1.1 = ($100 × 1.1) × 1.1 = $100 × 1.1² = $121</td>
</tr>
</tbody>
</table>

Interest is accrued on interest; hence, the name *compounded*!
The Compounding Process

• Assume you want to deposit $100 in a bank that offers a 10% interest rate that is *compounded annually*

  • What would be your total amount of money after 3 years?

<table>
<thead>
<tr>
<th>Year</th>
<th>Your Money</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>$100</td>
</tr>
<tr>
<td>1</td>
<td>$100 + ($100×0.1) = $100 × (1+0.1) = $100 × 1.1 = $110</td>
</tr>
<tr>
<td>2</td>
<td>$110 × 1.1 = ($100 × 1.1) × 1.1 = $100 × 1.1² = $121</td>
</tr>
<tr>
<td>3</td>
<td>$121 × 1.1 = (($100 × 1.1) × 1.1) × 1.1 = $100 × 1.1³ = 133.1</td>
</tr>
</tbody>
</table>
The Compounding Process

- In general, your initial capital will become:
  - \( a = a \times (1+i/100)^n \), where:
    - \( a \) is your initial capital
    - \( i \) is the interest rate as a percentage
    - And, \( n \) is the number of years
The Compounding Algorithm: Version 1

1. Input: a (initial capital)
2. Input: i (interest rate – in percentage)
3. Input: n (number of years)
4. \( j = 1 + \frac{i}{100} \)
5. \( c = a \times j^n \)
6. Output (the answer): c
The Compounding Algorithm: Version 2

1. Input: a (initial capital)
2. Input: i (interest rate – in percentage)
3. Input: n (number of years)
4. \[ c = a \times (1 + \frac{i}{100})^n \]
5. Output (the answer): c
Moving to Programming...

• Let us translate the compounding algorithm into a program using Python

• But, what is a program?
  • A program is just a sequence of instructions telling the computer what to do
  • These instructions need to be written in a language that computers can understand
  • This kind of a language is referred to as a programming language
  • Python is an example of a programming language

• Every structure in a programming language has an exact form (i.e., syntax) and a precise meaning (i.e., semantic)
Integrated Development Environment

• A special type of software known as a *Integrated Development Environment (IDE)* simplifies the process of writing (or *developing*) programs

• In this course, we will use an IDE named **Spyder**
  • It comes with **Anaconda**, a free and open-source distribution of Python for scientific computing (data science, machine learning applications, etc.,)
  • Let us download Anaconda and familiarize ourselves with Spyder
• Here is a very simple Python program:

```python
print("Hello")
print("Programming is fun!")
print(3)
print(2.3)
```

• `print(...)` is a built-in function that allows displaying information on screen
• When you call (or invoke) the print function, the parameters in the parentheses tell the function what to print
• There is only one parameter passed to the print function here, which is either a textual data (or what is denoted as a string like “Hello”), or integer (e.g., 3), or float (e.g., 2.3)
Simple Assignment Statements

• We can also define *variables* and assign them *values*

  a. *x* is a variable and 2 is its value

  b. *x* can be assigned different values; hence, it is called a variable

Output:

```
x = 2
x = 2.3
print(x)
```

```
Output:
2.3
```
In Python, values may end up anywhere in computer memory, and variables are used to refer to them.

Before

\[ x = 2 \]
\[ x = 2.3 \]

After

\[ x = 2.3 \]

What will happen to value 2?
Garbage Collection

• Interestingly, as a Python programmer you do not have to worry about computer memory getting filled up with old values when new values are assigned to variables.

• Python will automatically clear old values out of memory in a process known as **garbage collection**.

Memory location will be automatically reclaimed by the garbage collector.
Variable Names

- Python has some rules about how variable names can be written
  - Every variable name must begin with a letter or underscore, which may be followed by any sequence of letters, digits, or underscores

```
x1 = 10
x2 = 20
y_effect = 1.5
celsius = 32
2celsius = 2
```
Variable Names

• Python has some rules about how variable names can be written
  • Variable names are also *case-sensitive*

```python
x = 10
X = 5.7
print(x)
print(X)
```

**Output:**

```
10
5.7
```
Variable Names

- Python has some rules about how variable names can be written
  - Some names are part of Python itself (they are called *reserved words* or *keywords*) and cannot be used by programmers as ordinary names

<table>
<thead>
<tr>
<th>False</th>
<th>class</th>
<th>finally</th>
<th>is</th>
<th>return</th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
<td>continue</td>
<td>for</td>
<td>lambda</td>
<td>try</td>
</tr>
<tr>
<td>True</td>
<td>def</td>
<td>from</td>
<td>nonlocal</td>
<td>while</td>
</tr>
<tr>
<td>and</td>
<td>del</td>
<td>global</td>
<td>not</td>
<td>with</td>
</tr>
<tr>
<td>as</td>
<td>elif</td>
<td>if</td>
<td>or</td>
<td>yield</td>
</tr>
<tr>
<td>assert</td>
<td>else</td>
<td>import</td>
<td>pass</td>
<td></td>
</tr>
<tr>
<td>break</td>
<td>except</td>
<td>in</td>
<td>raise</td>
<td></td>
</tr>
</tbody>
</table>

Python Keywords
Variable Names

• Python has some rules about how variable names can be written
  • Some names are part of Python itself (they are called *reserved words* or *keywords*) and cannot be used by programmers as ordinary names

An example...

```python
for = 4
```

*SyntaxError: invalid syntax*  
*A variable name cannot be a Python keyword*
You can produce new data (numeric or text) values in your program using *expressions*.

This is an expression that uses the *addition operator*:

```
x = 2 + 3
print(x)
```

This is another expression that uses the *multiplication operator*:

```
print(5 * 7)
```

This is yet another expression that uses the *addition operator* but to *concatenate* (or glue) strings together:

```
print("5" + "7")
```
Expressions

• You can produce new data (numeric, text, ...) values in your program using *expressions*

```python
x = 6
y = 2
print(x - y)
print(x / y)
print(x // y)
print(abs(-x))
```

**Another example...**

**Yet another example...**

```python
print(x * y)
print(x ** y)
print(x % y)
print(abs(-x))
```
## Expressions: Summary of Operators

<table>
<thead>
<tr>
<th>Operator</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>+</td>
<td>Addition</td>
</tr>
<tr>
<td>-</td>
<td>Subtraction</td>
</tr>
<tr>
<td>*</td>
<td>Multiplication</td>
</tr>
<tr>
<td>/</td>
<td>Float Division</td>
</tr>
<tr>
<td>**</td>
<td>Exponentiation</td>
</tr>
<tr>
<td>abs()</td>
<td>Absolute Value</td>
</tr>
<tr>
<td>//</td>
<td>Integer Division</td>
</tr>
<tr>
<td>%</td>
<td>Modulo</td>
</tr>
</tbody>
</table>

**Python Built-In Numeric Operations**
Functions

• Python allows putting a sequence of instructions (or statements) together to create a brand-new command or function.

```python
def hello():
    print("Hello")
    print("Programming is fun!")
```

These *indentations* are necessary to indicate that these two statements belong to the same *scope* or *block of code*, which belongs to this function.
Python allows putting a sequence of instructions (or statements) together to create a brand-new command or function. 

a. The first indentation is mandatory (not providing it will cause a syntax error) 

b. If the second indentation is not provided, print(“Programming is fun!”) will not be considered part of the hello() function, but rather an independent statement

```python
def hello():
    print("Hello")
    print("Programming is fun!")
```
Calling Functions

• After defining a function, you can call (or *invoke*) it by typing its name followed by parentheses

a. This is how we invoke our defined function `hello()`

b. Notice that the two print statements (which form one code block) were executed in sequence

def hello():
    print("Hello")
    print("Programming is fun!")

hello()

Output:
Hello
Programming is fun!
The Compounding Algorithm in Python

def compoundInterest(a, i, n):
    c = a * (1+i/100) ** n
    print(c)

compoundInterest(100, 10, 4)

Output:

146.41000000000008
Now that you know how to translate algorithms into code, translate the following sequence of instructions or steps into Python:

1. Start with the number 7
2. Multiply by the current month
3. Subtract 1
4. Multiply by 13
5. Add today’s day
6. Add 3
7. Multiply by 11
8. Subtract the current month
9. Subtract the current day
10. Divide by 10
11. Add 11
12. Divide by 100
Next Class...

• Recitation + Quiz