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

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…

Wedding Ring

- 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

Item	Price (QAR)	Calories
Muffin	6	480
Croissant	7	595
Chips	10	950
Hamburger	8	800
Chocolate	2	300
Fruit salad	5	200

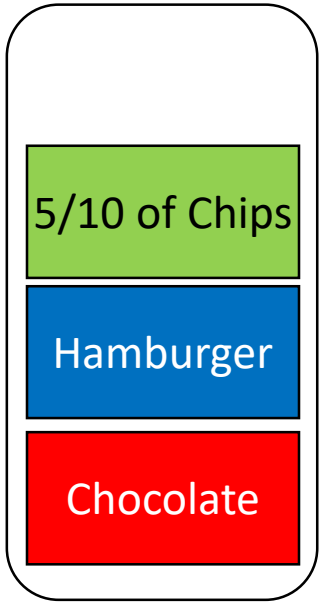
Objective: *Maximize* calories without exceeding a certain budget (*constraint*)

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

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



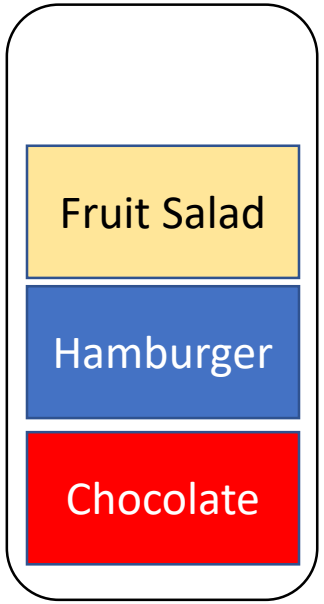
5/10 * 950 Cal	5 QAR
+	+
800 Cal	8 QAR
+	+
300 Cal	2 QAR
1,575 Cal	**15 QAR**

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**

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

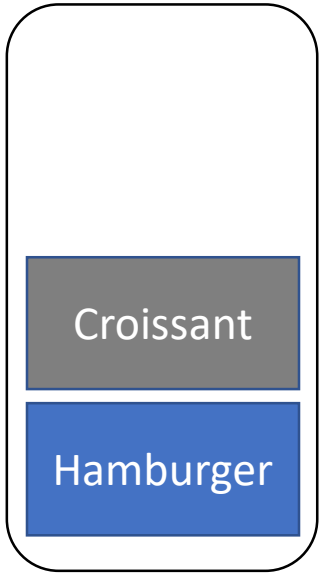


200 Cal	5 QAR
+	+
800 Cal	8 QAR
+	+
300 Cal	2 QAR
1,300 Cal	**15 QAR**

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

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



595 Cal	7 QAR
+	+
800 Cal	8 QAR
1,395 Cal	**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

Item	Weight (Kg)	Value (QAR)
Sceptre	4	10
Shoes	1	1
Helmet	1	2
Armour	12	4
Dagger	2	2

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?

Year	Your Money
0	$100
1	$100 + ($100×0.1) = $100 × (1+0.1) = $100 × 1.1 = $110
2	$110 × 1.1 = ($100 × 1.1) × 1.1 = $100 × 1.1^2 = $121

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?

Year	Your Money
0	$100
1	$100 + ($100×0.1) = $100 × (1+0.1) = $100 × 1.1 = $110
2	$110 × 1.1 = ($100 × 1.1) × 1.1 = $100 × 1.1^2 = $121
3	$121 × 1.1 = (($100 × 1.1) × 1.1) × 1.1 = $100 × 1.1^3 = 133.1

100×1.1^n

The Compounding Process

- In general, your initial capital will become:
  - $\mathbf{a} = \mathbf{a} \times (1+\mathbf{i}/100)^{\mathbf{n}}$, 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. $j = 1 + i/100$
5. $c = a \times j^n$
6. Output (the answer): c

$$\mathbf{a} = \mathbf{a} \times (1 + \mathbf{i}/100)^{\mathbf{n}}$$

The Compounding Algorithm: Version 2

1. Input: a (initial capital)
2. Input: i (interest rate – in percentage)
3. Input: n (number of years)
4. $c = a \times (1+i/100)^n$
5. Output (the answer): c

$$\mathbf{a} = \mathbf{a} \times (1+\mathbf{i}/100)^{\mathbf{n}}$$

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

Writing Python Programs

- Here is a very simple Python program:

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

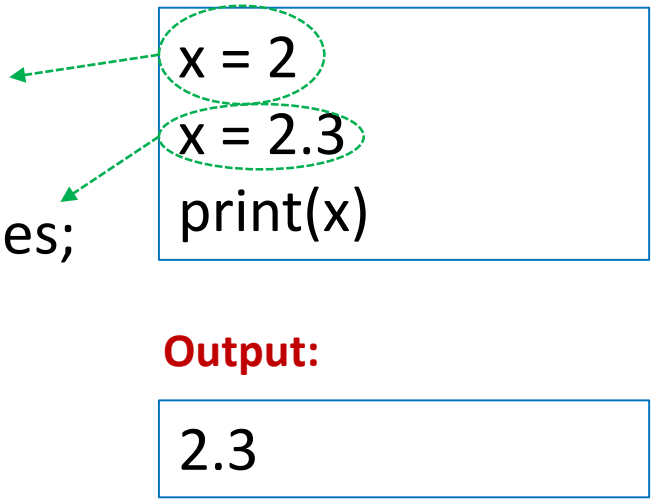
  - *print(…)* is a built-in *function* that allows displaying information on screen
  - When you call (or invoke) the print function, the *parameters* in the parentheses tell the function what to print
  - There is only one parameter passed to the print function here, which is either a *textual data (*or 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

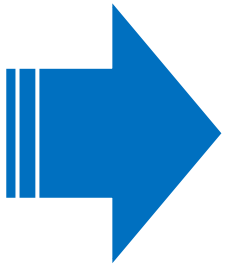
b. x can be assigned different values; hence, it is called a variable

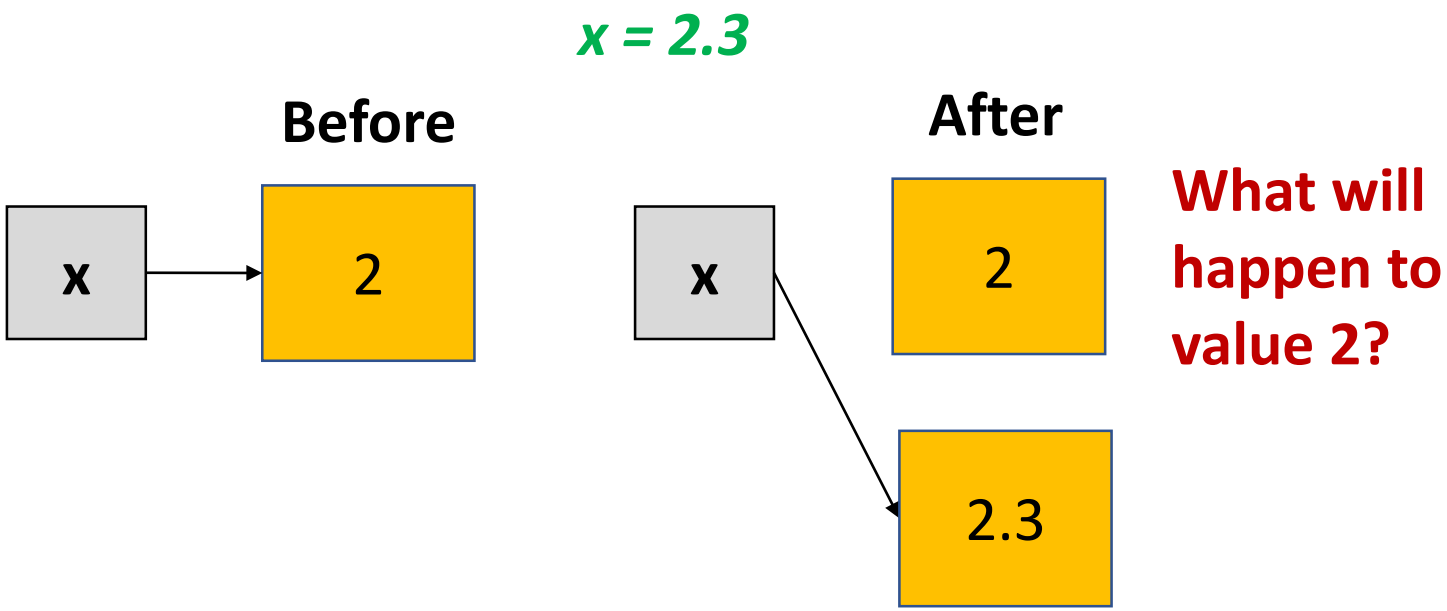


Simple Assignment Statements

- In Python, values may end up anywhere in computer *memory*, and variables are used to refer to them

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

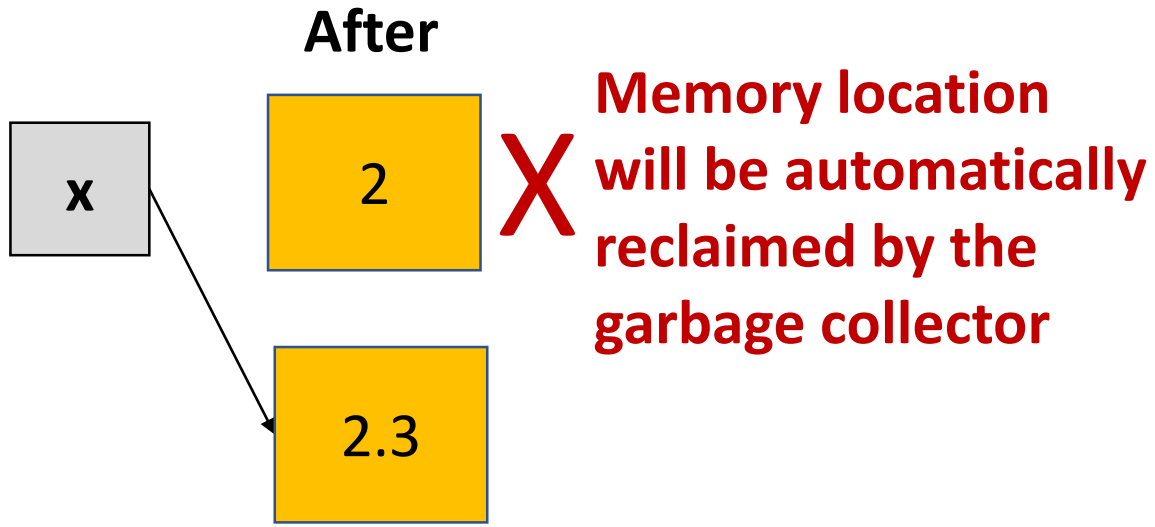




Garbage Collection

- 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*



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
```

SyntaxError: invalid syntax

A varilable name cannot start with a digit

Variable Names

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

```
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

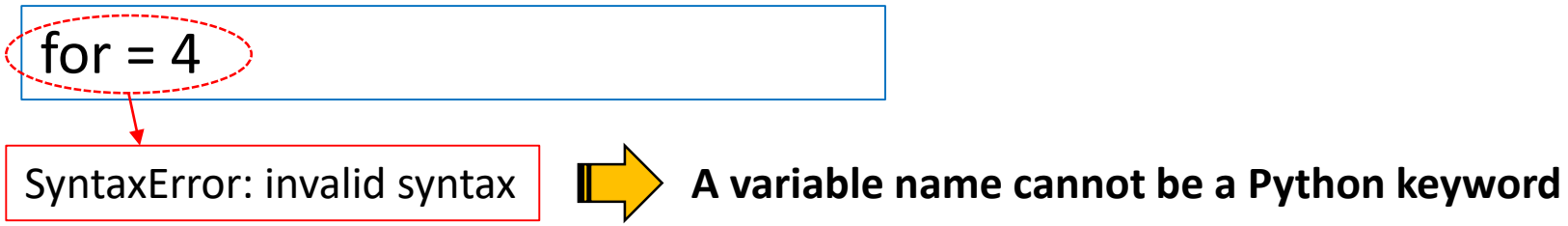
False	class	finally	is	return
None	continue	for	lambda	try
True	def	from	nonlocal	while
and	del	global	not	with
as	elif	if	or	yield
assert	else	import	pass	
break	except	in	raise	

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…



A variable name cannot be a Python keyword

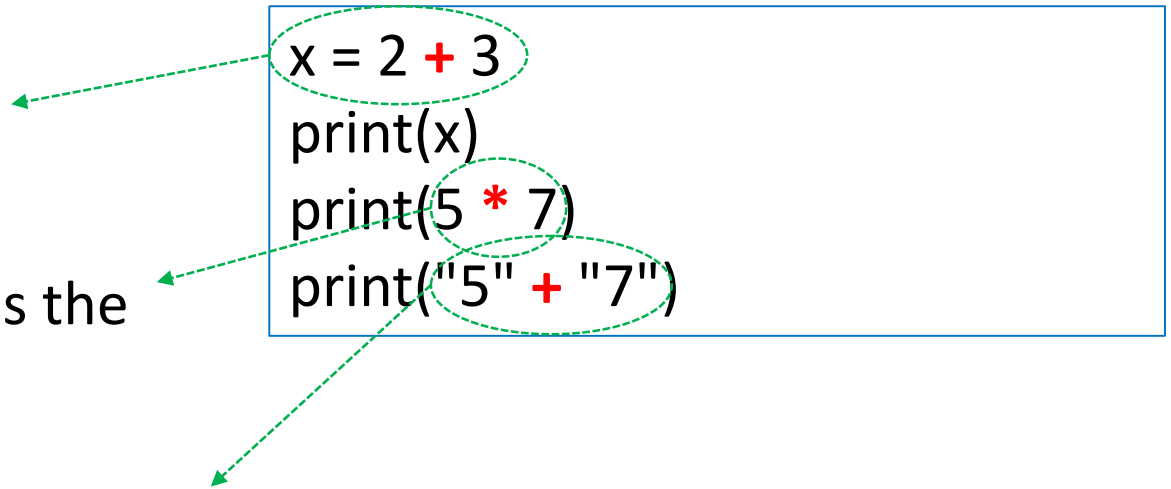
Expressions

- You can produce new data (numeric or text) values in your program using *expressions*

This is an expression that uses the *addition operator*

This is another expression that uses the *multiplication operator*

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



Expressions

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

Another example…

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

Yet another example…

```
print(x*y)
print(x**y)
print(x%y)
print(abs(-x))
```

Expressions: Summary of Operators

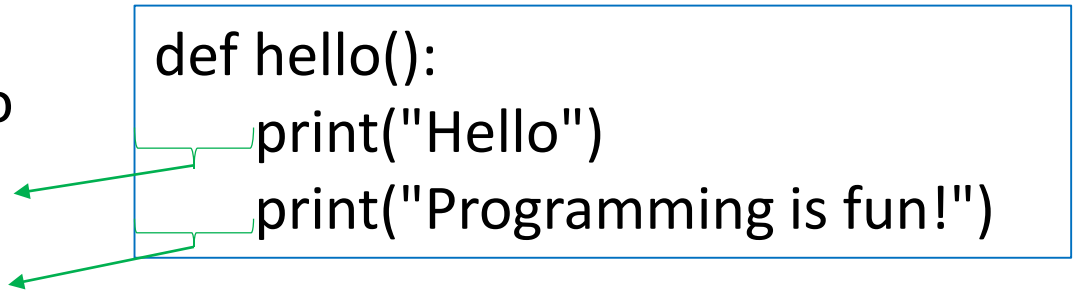
Operator	Operation
+	Addition
-	Subtraction
*	Multiplication
/	Float Division
**	Exponentiation
abs()	Absolute Value
//	Integer Division
%	Modulo

Python Built-In Numeric Operations

Functions

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

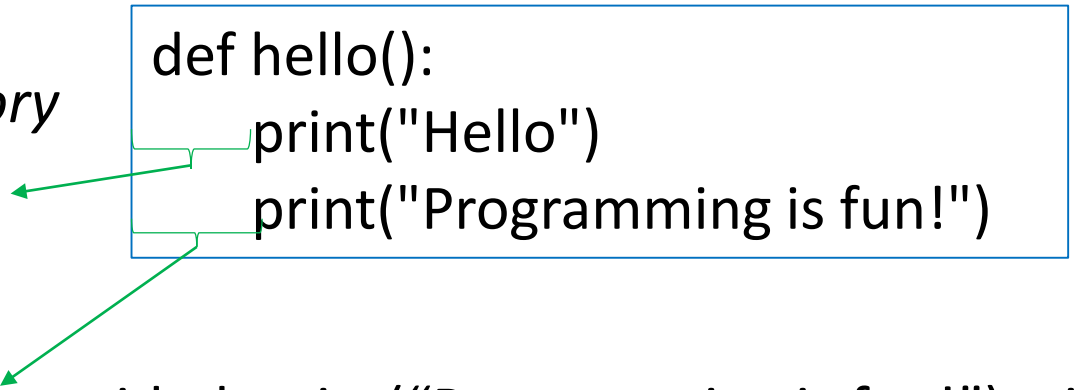
These *indentations* are necessary to indicate that these two statements belong to the same *scope* or *block of code*, which belongs to this function



Functions

- 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

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
```

Exercise

- 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