CSC 111: Intro to Computer Science through Programming

Spring 2017
Prof. Sara Mathieson
Admin

+ Homework 7 is due April 4 (tomorrow)

+ **Office hours today 3-5pm** (Ford 355, usually move to 345)

+ Thursday office hours moved to 10am-12pm for the rest of the semester

+ **Notecard**: please fill out and return to box (anonymous feedback)
  1) Something you understand well
  2) Concept that needs most work/review (“muddiest point”)
  3) Any other feedback?
Examples from last week
Midterm Part 5
Begin: recursion (Chap 13.1-13.2)
Friday: liberal arts module (maps)
Examples from last week
While loop for binary (Lab 7)

```python
# CSC 111 Lab 7
# Elise Snoey and Karen Santamaria
# Part C!

def binary(integer):
    string = ""
    while integer >= 1:
        remainder = integer % 2
        integer = integer // 2
        string = str(remainder) + string
    return string  # changed to return
```

Also implemented by:

- Garcia and Zoraida
- Christa and Megan
- Isabelle and Maggie
- Jess and Yingchuan
- Ruth
- Michelle and Caitlin
Christa
Midterm Part 5
In class we have seen how to swap the values of two variables, and in homework we have seen how to swap the values of two elements in a list. In this question, the goal is to swap the values of three variables, so that each variable ends up with the value of the “previous” variable. For example, if \( x = 6, \ y = 3, \) and \( z = 1, \) then the end result should be \( x = 1, \ y = 6, \) and \( z = 3. \)

(a) The following code shows a first attempt at this process. Fill in the table below, showing what will happen after each line is executed. The first row has been filled in with the initial values from the example above.
In class we have seen how to swap the values of two variables, and in homework we have seen how to swap the values of two elements in a list. In this question, the goal is to swap the values of three variables, so that each variable ends up with the value of the “previous” variable. For example, if $x = 6$, $y = 3$, and $z = 1$, then the end result should be $x = 1$, $y = 6$, and $z = 3$.

(a) The following code shows a first attempt at this process. Fill in the table below, showing what will happen after each line is executed. The first row has been filled in with the initial values from the example above.

<table>
<thead>
<tr>
<th>code</th>
<th>x</th>
<th>y</th>
<th>z</th>
<th>temp1</th>
<th>temp2</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>6</td>
<td>3</td>
<td>1</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>$\text{temp1} = x$</td>
<td>6</td>
<td>3</td>
<td>1</td>
<td>6</td>
<td>-</td>
</tr>
<tr>
<td>$\text{temp2} = y$</td>
<td>6</td>
<td>3</td>
<td>1</td>
<td>6</td>
<td>3</td>
</tr>
<tr>
<td>$x = z$</td>
<td>1</td>
<td>3</td>
<td>1</td>
<td>6</td>
<td>3</td>
</tr>
<tr>
<td>$y = x$</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>6</td>
<td>3</td>
</tr>
<tr>
<td>$z = y$</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>6</td>
<td>3</td>
</tr>
</tbody>
</table>
(b) Explain the issue with the code above, and rewrite the code with a few modifications so that it successfully swaps these three variables.
(b) Explain the issue with the code above, and rewrite the code with a few modifications so that it successfully swaps these three variables.

<table>
<thead>
<tr>
<th>code</th>
<th>x</th>
<th>y</th>
<th>z</th>
<th>temp1</th>
<th>temp2</th>
</tr>
</thead>
<tbody>
<tr>
<td>temp1 = x</td>
<td>6</td>
<td>3</td>
<td>1</td>
<td>6</td>
<td>-</td>
</tr>
<tr>
<td>temp2 = y</td>
<td>6</td>
<td>3</td>
<td>1</td>
<td>6</td>
<td>3</td>
</tr>
<tr>
<td>x = z</td>
<td>1</td>
<td>3</td>
<td>1</td>
<td>6</td>
<td>3</td>
</tr>
<tr>
<td>y = temp1</td>
<td>1</td>
<td>6</td>
<td>1</td>
<td>6</td>
<td>3</td>
</tr>
<tr>
<td>z = temp2</td>
<td>1</td>
<td>6</td>
<td>3</td>
<td>6</td>
<td>3</td>
</tr>
</tbody>
</table>
(c) Can you perform this three-variable swap with only one temporary variable? If no, explain the issue with such an approach. If yes, provide the code and a table like the one above.
(c) Can you perform this three-variable swap with only one temporary variable? If no, explain the issue with such an approach. If yes, provide the code and a table like the one above.
(d) Would it be possible to write a function to swap three variables? i.e. would it be possible to write a function that produces the output below? Explain your answer and reasoning, using the concept of mutable vs. immutable types.

```python
>>> x = 6
>>> y = 3
>>> z = 1
>>> three_way_swap(x, y, z)
>>> x
1
>>> y
6
>>> z
3
```
Part 5

(d) Would it be possible to write a function to swap three variables? i.e. would it be possible to write a function that produces the output below? Explain your answer and reasoning, using the concept of mutable vs. immutable types.

```python
>>> x = 6
>>> y = 3
>>> z = 1
>>> three_way_swap(x, y, z)
>>> x
1
>>> y
6
>>> z
3

Not possible! Ints are immutable.
(e) Write a function that will return a new string with the \textit{i}^{th} \ character and \textit{j}^{th} \ character swapped. Example: \texttt{string\_swap("spring",1,5)} should return \texttt{"sgrinp"}. You may assume that \textit{i} is less than \textit{j}. Why must this function return a \textit{new} string?
(e) Write a function that will return a new string with the \( i \)\textsuperscript{th} character and \( j \)\textsuperscript{th} character swapped. Example: `string_swap("spring",1,5)` should return "sgrinp". You may assume that \( i \) is less than \( j \). Why must this function return a new string?

```python
def string_swap(string, i, j):
    # Note that this will NOT work because strings are not mutable.
    temp = string[i]
    string[i] = string[j]
    string[j] = temp'

    before = string[:i]  # part before the \( i \)th character
    middle = string[i+1:j]  # part between the \( i \)th and \( j \)th characters
    after = string[j+1:]  # part after the \( j \)th character

    # swap the \( i \)th and \( j \)th characters
    new_string = before + string[j] + middle + string[i] + after
    return new_string  # return!
```
Begin: recursive functions
What is recursion?

- A recursive function calls itself.

- Usually two key components to a recursive function:
  1) A base case or way to stop the recursion
  2) A recursive call using a modified argument
What is recursion?

- A recursive function calls **itself**.

- Usually two key components to a recursive function:
  1) A **base case** or way to stop the recursion
  2) A **recursive call** using a modified argument

“To understand recursion, you must first understand recursion”
# Factorial function

The factorial of a non-negative integer $n$, denoted by $n!$, is the product of all positive integers less than or equal to $n$.

$n! = n \times (n-1) \times (n-2) \times \ldots \times 3 \times 2 \times 1$

or

$n! = n \times (n-1)!$

with

base case: $0! = 1$

<table>
<thead>
<tr>
<th>$n$</th>
<th>recursion</th>
<th>$n!$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>base case</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## Factorial function

The factorial function, denoted as $n!$, is defined as the product of all positive integers less than or equal to $n$. It can be recursively defined as follows:

1. **Base Case**: $0! = 1$
2. **Recursion**: $n! = n \times (n-1)!$ for $n > 0$

### Table of Factorials

<table>
<thead>
<tr>
<th>$n$</th>
<th>recursion</th>
<th>$n!$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>base case</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>$1 \times 1$</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

$n! = n \times (n-1) \times (n-2) \times \ldots \times 3 \times 2 \times 1$

or

$n! = n \times (n-1)!$

with

base case: $0! = 1$
Factorial function

The factorial function, denoted as \( n! \), is defined as:

- **Base case:** \( 0! = 1 \)
- **Recursion:** \( n! = n \times (n-1)! \)
  
\[
\begin{array}{|c|c|c|}
\hline
n & \text{recursion} & n! \\
\hline
0 & \text{base case} & 1 \\
1 & 1 \times 1 & 1 \\
2 & 2 \times 1 & 2 \\
3 & & \\
4 & & \\
5 & & \\
6 & & \\
\hline
\end{array}
\]

\[
n! = n \times (n-1) \times (n-2) \ldots 3 \times 2 \times 1
\]

or

\[
n! = n \times (n-1)!
\]

with base case: \( 0! = 1 \)
Factorial function

\[ n! = n \times (n-1) \times (n-2) \times \ldots \times 3 \times 2 \times 1 \]

or

\[ n! = n \times (n-1)! \]

with

base case: \( 0! = 1 \)
**Factorial function**

\[ n! = n \times (n-1) \times (n-2) \cdots 3 \times 2 \times 1 \]

or

\[ n! = n \times (n-1)! \]

with

base case: \( 0! = 1 \)

<table>
<thead>
<tr>
<th>n</th>
<th>recursion</th>
<th>n!</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>base case</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1*1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>2*1</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>3*2</td>
<td>6</td>
</tr>
<tr>
<td>4</td>
<td>4*6</td>
<td>24</td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Factorial function

\[
n! = n \times (n-1) \times (n-2) \times \cdots \times 3 \times 2 \times 1
\]

or

\[
n! = n \times (n-1)!
\]

with

base case: \(0! = 1\)

<table>
<thead>
<tr>
<th>(n)</th>
<th>recursion</th>
<th>(n!)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>base case</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1*1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>2*1</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>3*2</td>
<td>6</td>
</tr>
<tr>
<td>4</td>
<td>4*6</td>
<td>24</td>
</tr>
<tr>
<td>5</td>
<td>5*24</td>
<td>120</td>
</tr>
<tr>
<td>6</td>
<td>[\text{not shown}]</td>
<td>[\text{not shown}]</td>
</tr>
</tbody>
</table>
Factorial function

\[ n! = n \times (n-1) \times (n-2) \times \ldots \times 3 \times 2 \times 1 \]

or

\[ n! = n \times (n-1)! \]

with base case: 0! = 1

<table>
<thead>
<tr>
<th>n</th>
<th>recursion</th>
<th>n!</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>base case</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1\times1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>2\times1</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>3\times2</td>
<td>6</td>
</tr>
<tr>
<td>4</td>
<td>4\times6</td>
<td>24</td>
</tr>
<tr>
<td>5</td>
<td>5\times24</td>
<td>120</td>
</tr>
<tr>
<td>6</td>
<td>6\times120</td>
<td>720</td>
</tr>
</tbody>
</table>