CMPT 120
Intro to CS & Programming I

WEEK 13 (Apr. 7-Apr. 11)

— *Jérémie O. Lumbroso* —

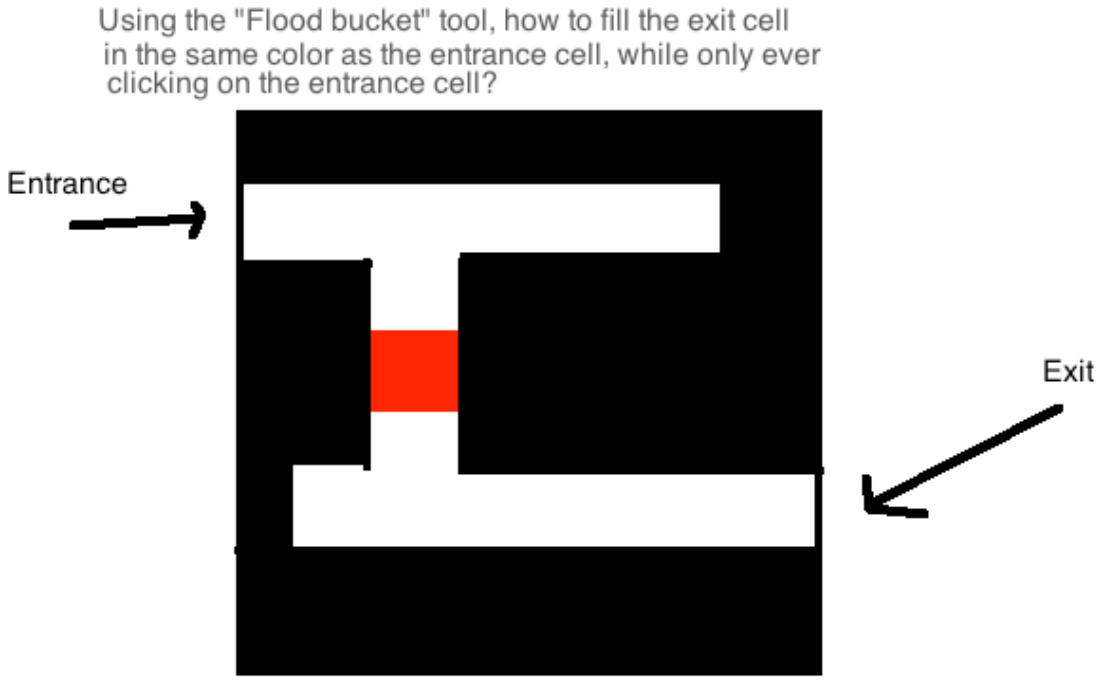
Lecture 31:
The Recap Lecture

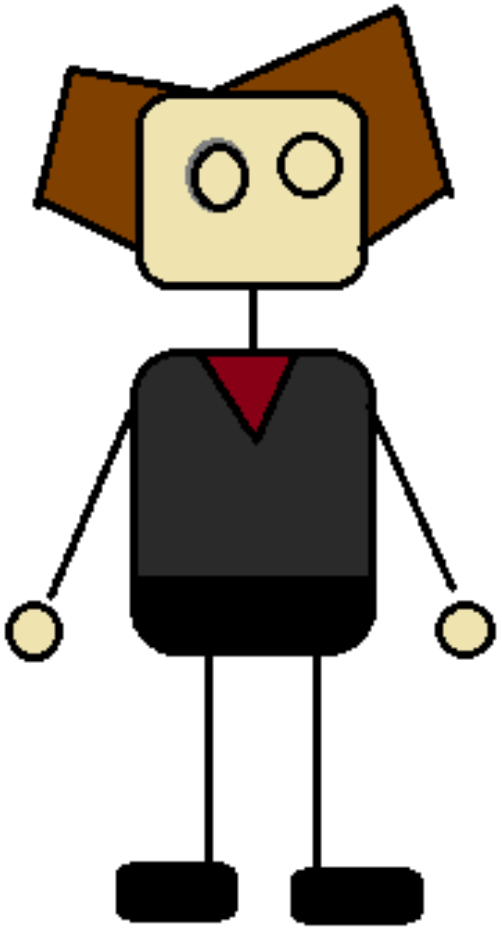
http://www.sfu.ca/~jlumbros/Courses/CMPT120/

PRACTICE QUESTION 4

Assignment 2

- Hint with the minotaur
- (Minor question: *why do you guys have trouble spelling this word?*)





DOES THIS LOOK LIKE ME???

A Yes, what an incredible likeness!

B Yes, I want a T-shirt with you on it!

C Yes, you moron are even wearing this today

D I hate you

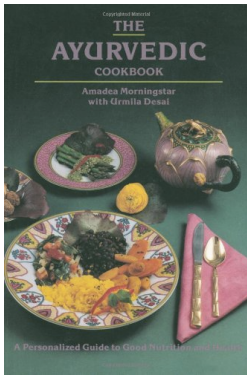
E I understood you are about the grade my final exam and that you probably are not pleased by this crude caricature, so I am going to say: **NO, SIR, THIS DOES NOT CAPTURE YOUR INCREDIBILITY**

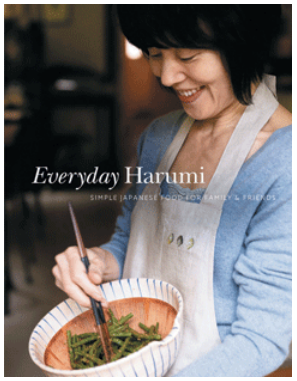
From algorithms, to pseudocode, to Python, to variables, to integers

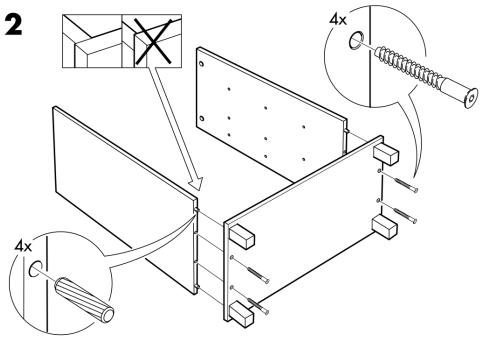
THEORY & ELEMENTARY PYTHON

From Algorithms to Programming

- An algorithm: sequence of unambiguous instructions to solve a problem in a finite amount of time
- Cooking recipe, or IKEA building instructions







- Algorithms for computers: given with programming languages
- Most programming languages are equally powerful: Python, Java, C/C++, Basic, Go, etc.

Pseudo-Code

- Pseudocode is sometimes an easier way – an intermediate step to describe algorithms, before programming them
- Flood-fill in pseucode
  - for every cell *x* of the maze
    - if any of the neighboring cells of is colored
    - and if x is an empty cell
      - then color the cell x
- Translation to code in next slide

Flood Fill in Code

```
for i in range(len(grid)):
  for j in range(len(grid[0])):
    # For every cell:
    ## Does the cell exist? ## Is it colored?
    if (i+1 < len(grid)    and grid[i+1][j] == COLORED) or
       (i-1 >= 0           and grid[i-1][j] == COLORED) or
       (j+1 < len(grid[0]) and grid[i][j+1] == COLORED) or
       (j-1 >= 0           and grid[i][j-1] == COLORED):
      if grid[i][j] == EMPTY:
        grid[i][j] == COLORED
```

Variables

- **Variable:** storage and associated name
- Assignment to variable using single equal sign
- Variables have a **type**: integer, float, string, list, boolean (True/False value)

```
myvar = "jeeny"
if type(myvar) == str:
   print "myvar contains a string"
```

- Variables cannot be referenced (their value cannot be used) **unless it has been initialized**: i.e., assigned to at least once

Variable Scope

- A given variable is not always accessible from everywhere
- Though we can use **global** variables, in general
  - a variable defined inside a function cannot be accessed anywhere else
  - variables defined outside of a function (in the top-level – the code with no indentation) can be read anywhere but can only be modified in the top-level

Integers and Floats

- Python has several types to store numbers
- The normal math operations work, with the expected rules: +, -, *, / and exponentiation is **
- Division between integers rounds down to the nearest integer, i.e., `5/4 == 1` not 2; you must use at least one float to get a more precise result
- Conversion to integers is done with `int(...)`, to floats with `float(...)`

If statements and for loops

ELEMENTARY CONSTRUCTIONS

`if` statement

- When have to adapt our actions depending on a condition, we use an if statement:

```
if <condition(s)>:
  print "The condition is true"
else:
  print "The condition is false"
```

- When testing for **equality**, one must use the double equal sign == (or != for inequality)
- Multiple conditions can be combined using logical operators: **and**, **or**, and **not**

```
if (not sick and students == nice) or instructor == moron:
  print "Instructor will teach whole day tutorial"
```

`if` and `else` and `elif`

- An if statement can either be alone

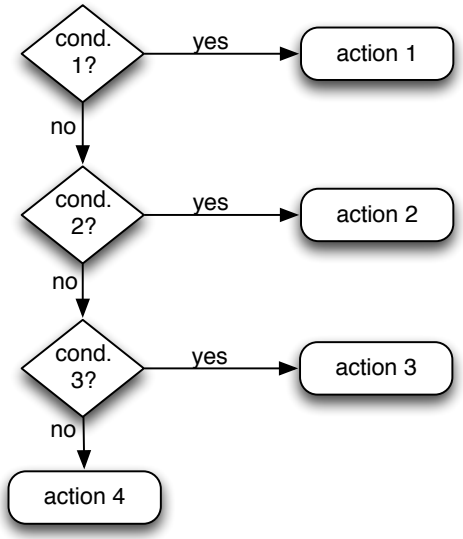
```
if age >= legal:
    print "Let's get it on…"
```

- Or it can contain what we call ***alternatives***
- Either just a default case using an `else` (what happens if the condition is not verified)
- Or different conditions to be tested in order with `elif` (short for "else if")

`elif` versus `if`

if-elif-elif-else

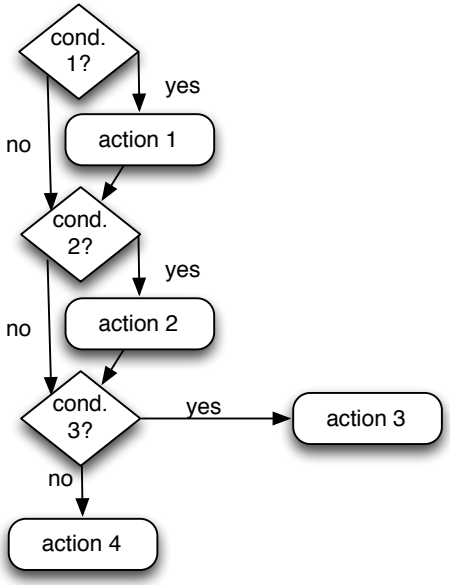
```
if <cond 1>:
  <action 1>
elif <cond 2>:
  <action 2>
elif <cond 3>:
  <action 3>
else:
  <action 4>
```



- Only one action is executed
- **Conditions tried in order**

if-if-if-else

```
if <cond 1>:
  <action 1>
if <cond 2>:
  <action 2>
if <cond 3>:
  <action 3>
else:
  <action 4>
```



- With if/if/…, possibly as many actions as if blocks

`for` loops

- The `for` loop allows you to iterate over a *finite* range of numbers; this will print numbers from 1 to 9 (`range(n)` goes up to `n-1`)

```
for i in range(1,10):
    print i
```

- Block underneath `for` loop is repeated as many times as there are elements in `range(1, 10)`
- Each iteration, i takes different value from range

Iterating over Strings and Lists

- Iteration with a for loop works for more than just ranges
- It also works for
  - characters in a string s: `for ch in s: ...`
  - elements in a list L: `for elt in L: ...`
- The syntax is always the same

Strings

- Strings are specified in Python using single quotes or doubles quotes:

```
"this is a string"
'this is also a string'
this is not a string
```

- **VERY CAREFUL:** if you do not use quotes, then you are referencing variables names:
- `"hello"` # this is the string "hello"
- `hello` # this is a variable called hello, that is not initialized for the moment

Common String Operations

- String operations and list operations are the same
- `sA = "hello"  sB = " juanita"`

operation	syntax	meaning (+ example)
indexing	`[ ]`	access an character of a string (or element of a list) `sA[2]` gives `"l"`
concatenation	`+`	combine strings/lists together `sA + sB` gives `"hello juanita"`
repetition	`*`	repeats string/list multiple times `"aa" * 4` gives `"aaaaaaaa"`
membership	`in`	determine if a character is in a string determines if an element is in a list (but is VERY slow) `"e" in sA` gives `True`; `"a" in sA` returns `False`
length	`len`	gives the length of a string/list `len(sA)` gives `5` and `len(sB)` gives `7`
slicing	`[ : ]`	extracts a substring from a string extracts a sequence of elements from a list `sA[1:3]` gives `"el"` (elements from `1` to `3 - 1 = 2`)

Iteration on Strings

- There are two ways of iterating over strings

```
s = "hello"
```

- Iterate over the positions of the characters (you must then use the len(…) function)

```
for i in range(len(s)):
  print "char in position", i, "is", s[i]
```

This is important when you need to remember or know the position of a character (for instance when writing a find_pos function)

- Iterate over the characters

```
for ch in s:
  print "char", ch
```

A more complex form of iteration

WHILE LOOPS

`while` loop syntax

```
while <condition>:
    <actions>
```

- Every iteration, the loop executes `<actions>`
- The loop goes on as long as `<condition>` is verified
- The `<actions>` block **must** contain something to make the condition evolve, or else there is a risk a of **loop being infinite**

Examples of infinite while loops

Correct while loops

```
i = 0
while i < 4:
 print i
 i = i + 1
```

```
i = 4
while i > 5:
 print i
 i = i + 1
```

Incremental while loops are infinite if

- there is no increment (2 & 5)

```
i = 1
while i < 9:
 print i
```

```
i = 1
while i < 9:
 print i
 # i = i + 1
```

- the increment won't make condition change (4 & 6)

```
i = 10
while i > 2:
 print i
 i = i + 1
```

```
i = 4
while i < 7:
 print i
 i = i - 1
```

Be careful!

FINAL PACING QUESTION

How well did you understand today?

A. Too easy or **too slow**

B. Everything went at a good pace, and I am fine

C. Good riddance, I don't care if I understand so long as I'm done

D. I think I am going to die

E. I was drinking every word you spoke like one of the Ten Commandments, because I am crying inside knowing I will not have you teaching me this course again

Thank You For Being So Great

And Now My Heart Is Like This