Welcome to Programming Bootcamp!

(Please sign-in in the back)

Penn
University of Pennsylvania

Sponsored by:

Institute for Biomedical Informatics

BGSA

GCB

Lab1: Common mistakes

- ```
  a=3.1415926
  print'{:.2f}'.format(a)
  ```
- ```
  print "I have" + 5 + "avocados"
  ```
- ```
  print -4 ** 2
  ```
- ```
  x = "4"
  print (x * 3)
  ```
- ```
  x = 25
  print x ** 0.5
  ```
- ```
  age = "26"
  name = "Raju"
  introduction = "Hello, my name is " , name , " and I am " , age , " years old."
  print introduction
  ```

Please look out how to submit hwk files post.
Class assignment = hwk
Sorry for the confusion!

Writing code that makes decisions: if/else statements

Lesson 2 – 7/25/18

Slides created by: Dr. Sarah Middleton

Today's topics

1. `if/else` statements
2. Built-in functions
   - `input(),len(),abs(), round()`
3. Non-built-in functions
   - a brief intro to modules
4. How to read the PyDocs
5. Commenting your code

1. if/else statements

Control statements – what are they?

Programming is a lot like giving someone instructions or directions. For example, if I wanted to give you directions to my house, I might say...

- Turn right onto Main Street
- Turn left onto Maple Ave
- **If** there is construction, continue straight on Maple Ave, turn right on Cat Lane, and left on Fake Street; **otherwise** cut through the empty lot to Fake Street
- Go straight on Fake Street until house 123

Control statements – what are they?

The same directions, but in code:

```
construction = False
print("Turn right onto Main Street”)
print("Turn left onto Maple Ave”)
if construction:
          print("Continue straight on Maple Ave”)
          print("Turn right onto Cat Lane”)
          print("Turn left onto Fake Street”)
else:
          print("Cut through the empty lot to Fake Street”)
print ("Go straight on Fake Street until house 123”)
```

Output:

```
Turn right onto Main Street
Turn left onto Maple Ave
Cut through the empty lot to Fake Street
Go straight on Fake Street until house 123
```

Control statements – what are they?

The same directions, but in code:

```
construction = False
print("Turn right onto Main Street")
print("Turn left onto Maple Ave")
if construction:
        print("Continue straight on Maple Ave")
        print("Turn right onto Cat Lane")
        print("Turn left onto Fake Street")
else:
        print("Cut through the empty lot to Fake Street")
print("Go straight on Fake Street until house 123")
```

True and **False** are special words in Python called **Booleans**

This is called an "**if statement**". It works pretty much how you'd expect it to: if the statement is true, it executes the first block of code; if the statement is false, it executes the second block of code (under the `else`)

Output:

```
Turn right onto Main Street
Turn left onto Maple Ave
Cut through the empty lot to Fake Street
Go straight on Fake Street until house 123
```

Since `construction` holds the value **False**, the `if` statement skips the first block of print statements and only executes what is in the `else:` block.

Booleans - the logical datatype

- A Boolean ("bool") is actually a type of variable, like a string, int, or float. However, a Boolean is only allowed to take the values `True` or `False`.
- `True` and `False` are always capitalized and never in quotes.
- Don't think of `True` and `False` as words. You can't treat them like you would strings. To the computer, they're actually interpreted as the numbers 1 and 0, respectively.

`if/else` statement

Purpose: creates a "fork" in the flow of the program.

- Based on the Boolean value of a conditional statement, either executes the `if`-block or the `else`-block
- The "blocks" are indicated by indentation.
- The `else`-block is optional.

`if/else` statement

Syntax:

```
if conditional:
        this code is executed
else:
        this code is executed
```

Example:

```
x = 5
if (x > 0):
        print("x is positive”)
else:
        print("x is negative”)
```

Important to note:

- Colons are required after the `if` condition and after the `else`
- All code that is part of the `if/else` statement must be indented.

What kinds of "conditionals" are allowed?

Anything that can be evaluated as true or false!

- is `a` True?
- is `a` less than `b`?
- is `a` equal to `b`?
- is `a` equal to "ATGCTG"?
- is (`a` greater than `b`) and (`b` greater than `c`)?

Forming conditionals

We use a special set of symbols/words to test whether statements are true or false:

Symbol	Meaning	Example
`==`	is equal to	`if (a == 4):`
`!=`	is not equal to	`if (a != "applesauce"):`
`<`	is less than	`if (a < 10.5):`
`<=`	is less than or equal to	`if (a <= b):`
`>`	is greater than	`if (a > (b * 2)):`
`>=`	is greater than or equal to	`if (a >= -1):`
`and`	and	`if (a > 0) and (a != 3):`
`or`	or	`if (a > 5) or (a < -5):`
`not`	not	`if not (a == 1):`

Question 1

What will this code print?

```
a = True
if a:
   print ("Hooray, a was true!”)
```

Optional: enter your answers online!
https://goo.gl/forms/x1G57MV2OXJ8tXOG2

Question 1

What will this code print?

```
a = True
if a:
   print ("Hooray, a was true!”)
```

Result

```
Hooray, a was true!
```

Question 2

What will this code print?

```
a = True
if a:
   print ("Hooray, a was true!”)
print ("Goodbye now!”)
```

Question 2

What will this code print?

```
a = True
if a:
   print("Hooray, a was true!”)
print("Goodbye now!”)
```

Result

```
Hooray, a was true!
Goodbye now!
```

Question 3

What will this code print?

```
a = False
if a:
   print("Hooray, a was true!”)
print("Goodbye now!”)
```

Question 3

What will this code print?

```
a = False
if a:
   print("Hooray, a was true!”)
print("Goodbye now!”)
```

Result

```
Goodbye now!
```

Question 4

What will this code print?

```
morning = True
if morning:
    print("Good morning!”)
else:
    print("Hello!”)
print("How are you?”)
```

Question 4

What will this code print?

```
morning = True
if morning:
    (print "Good morning!”)
else:
    (print "Hello!”)
print("How are you?”)
```

Result

```
Good morning!
How are you?
```

Question 5

What will this code print?

```
a = True
b = False
if a and b:
    print("Apple”)
else:
    print("Banana”)
```

Question 5

What will this code print?

```
a = True
b = False
if a and b:
    print("Apple”)
else:
    print("Banana”)
```

Result

```
Banana
```

Question 6

What will this code print?

```
a = True
b = False
if a and not b:
    print("Apple”)
else:
    print("Banana”)
```

Question 6

What will this code print?

```
a = True
b = False
if a and not b:
    print("Apple”)
else:
    print("Banana”)
```

Result

```
Apple
```

Question 7

What will this code print?

```
a = True
b = False
if not a and b:
    print("Apple”)
else:
    print(”Banana”)
```

Question 7

What will this code print?

```
a = True
b = False
if not a and b:
    print("Apple”)
else:
    print("Banana”)
```

Result

```
Banana
```

Question 8

What will this code print?

```
a = True
b = False
if not (a and b):
    print("Apple”)
else:
    print("Banana”)
```

Question 8

What will this code print?

```
a = True
b = False
if not (a and b):
    print "Apple"
else:
    print "Banana"
```

Result

```
Apple
```

Question 9

What will this code print?

```
a = True
b = False
if a or b:
    print("Apple”)
else:
    print("Banana”)
```

Question 9

What will this code print?

```
a = True
b = False
if a or b:
    print("Apple”)
else:
    print("Banana”)
```

Result

```
Apple
```

Question 10

What will this code print?

```
a = True
b = False
if not (a or b):
    print("Apple”)
else:
    print("Banana”)
```

Question 10

What will this code print?

```
a = True
b = False
if not (a or b):
    print("Apple”)
else:
    print("Banana”)
```

Result

```
Banana
```

Question 11

What will this code print?

```
a = 5
b = 10
if (a == 5) and (b > 0):
    print "Apple"
else:
    print "Banana"
```

Question 11

What will this code print?

```
a = 5
b = 10
if (a == 5) and (b > 0):
    print "Apple"
else:
    print "Banana"
```

Result

```
Apple
```

Question 12

What will this code print?

```
a = 5
b = 10
if ((a == 1) and (b > 0)) or (b == (2 * a)):
    print "Apple"
else:
    print "Banana"
```

Question 12

What will this code print?

```
a = 5
b = 10
if ((a == 1) and (b > 0)) or (b == (2 * a)):
    print "Apple"
else:
    print "Banana"
```

Result

```
Apple
```

Note on indentation

- Indentation is very important in Python; it's how Python tells what code belongs to which control statements
- Consecutive lines of code with the same indenting are sometimes called "blocks"
- Indenting should only be done in specific circumstances (if statements are one example, and we'll see a few more soon). Indent anywhere else and you'll get an error.
- You can indent by however much you want, but you must be consistent. Pick one indentation scheme (e.g. 1 tab per indent level, or 4 spaces) and stick to it.
- NOTE: Indentation via tab key is different between Macs and Windows, so if you try to run the same file it may not work and you will have to re-indent all your lines.

Other forms of the `if` statement

Multi-`if/else`:

```
choice = input("Choose option 1, 2, or 3: ")
if (choice == "1"):
        print("You have chosen option 1: cake”)
elif (choice == "2"):
        print("You have chosen option 2: ice cream”)
elif (choice == "3"):
        print("You have chosen option 3: broccoli”)
else:
        print("Invalid input.”)
```

Only one of these code blocks will be executed.

Other forms of the `if` statement

"Nested" `if/else`:

```
test = input("What is 1+1? ")
if (test == "2"):
        print("Correct!”)
        test2 = input("What is 2314*32626? ")
        if (test2 == "75496564"):
                print("Correct! You passed all my tests!”)
        else:
                print("Sorry, that's wrong.”)
else:
        print("Sorry, that's wrong.”)
```

2. Built-in functions

What's a built-in function?

- Python provides some useful built-in functions that perform specific tasks
- What makes them "built-in"?
  - Simply that you don't have to "import" anything in order to use them -- they're always available
- We've already seen some examples:
  - `print`, `int()`, `float()`, `str()`
- Now we'll look at a few more

`input()`

Description: A built-in function that allows user input to be read from the terminal.

- formely `raw_input()` in python 2.7
- As seen in the lab1 problem set.
- The execution of the code will pause when it reaches the `input()` function and wait for the user to input something.
- The input ends when the user hits "enter".
- The data that is read by `input()` can then be stored in a variable and used in the code.
- **This function always returns a string, even if the user entered a number.**

This allows us to change what our program does without actually changing the code itself!

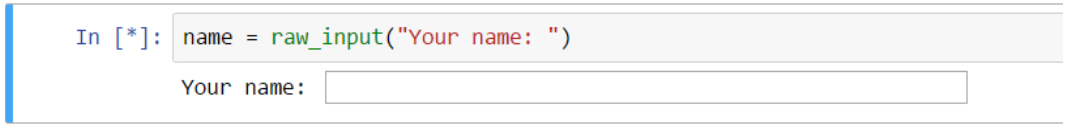
input()

Syntax:

```
input("Optional prompt: ")
```

Examples:

```
name = input("Your name: ")
age = int(input("Your age: "))
```



Important to note:

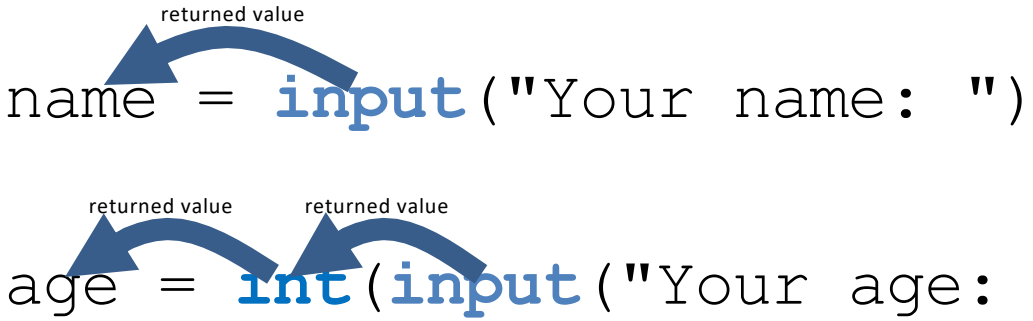
We can **nest commands** inside of each other, as in the second example here. This works a lot like the order of operations in math—whatever is the most nested is executed first, and then execution proceeds outward.

input()

Syntax:

```
input("Optional prompt: ")
```

Examples:



We say that `input()` "returns" a value (in this case, a string version of whatever was entered in the terminal)
This value can then be used by another function (e.g. `int()`) or saved in a variable.

`len()`

Description: Returns the length of a string (also works on certain data structures). Doesn't work on numerical types.

Examples:

```
print len("cat")
```

```
3
```

```
print len("hi there")
```

```
8
```

```
seqLength = len("ATGGTCGCAT")
```

returned value
saved in variable

abs()

Description: Returns the absolute value of a numerical value. Doesn't accept strings.

Examples:

```
print abs(-10)
```

```
10
```

```
print abs("-10")
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-7-441ce54f04bb> in <module>()
----> 1 print abs("-10")

TypeError: bad operand type for abs(): 'str'
```

```
print(int("-10"))
```

```
-10
```

```
positiveNum = abs(-23423)
```

round()

Description: Rounds a float to the indicated number of decimal places. If no number of decimal places is indicated, rounds to zero decimal places.

Synatx:

```
round(someNumber, numDecimalPlaces)
```

Examples:

```
print round(10.12345)
```

```
10.0
```

```
print round(10.12345, 2)
```

```
10.12
```

```
print round(10.9999, 2)
```

```
11.0
```

There are many more!

- Most of the other built-in functions are too advanced right now, but we'll see some more in the future
- If you're curious, there's a full list here:

https://docs.python.org/2/library/functions.html

3. Non-built-in functions

What is a non-built-in function?

- The only difference is that these functions aren't accessible until you **import** them
- Why aren't they all just built-in? It improves speed and memory usage to only import what is needed.
- Related functions are grouped into modules. Importing a module imports all the module's functions.
- We'll go over two modules today: `math` and `random`

How to use a module

1. First you must import the module. Add this to the top of your script:

```
import <moduleName>
```

2. To use a function of the module, you must prefix the function with the name of the module (using a period between them):

```
<moduleName>.<functionName>
```

The `math` module

Description: Contains many advanced math-related functions.

Usage example:

```
import math

math.sqrt(4)
math.log10(1000)
math.sin(1)
math.cos(0)
```

Important to note:

These functions all return values! (As do most functions)

You must save the returned value in a variable to use it elsewhere in the code.

The `random` module

Description: contains functions for generating random numbers.

Usage example:

```
import random


random.random()
random.randint(0,10)
random.gauss(5, 2)
```

Important to note:

These functions all return values! (As do most functions)

You must save the returned value in a variable to use it elsewhere in the code.

Variations of importing

- You can import more than one module at a time:

```
import math, random
```

- You can give a module an alias and use that in your code (good when module name is long):

```
import random as rnd
```

- You can import individual functions from a module (note that if you do this, you must call the function *WITHOUT* prefixing it):

```
from math import log10
log10(100)
```

- You can import all functions as above if you use * (so then you can use all functions without prefixing with the module name):

```
from math import *
sqrt(64)
```

Important to note:
This last one is generally a bad idea, actually. It can lead to a lot of confusion in large scripts with many modules in use. Use it sparingly, if at all.

There are many more!

If you are curious, there's a list of modules here:

https://docs.python.org/2.7/py-modindex.html

4. Understanding the PyDocs

A quick primer on reading the PyDocs

Example PyDoc entry:

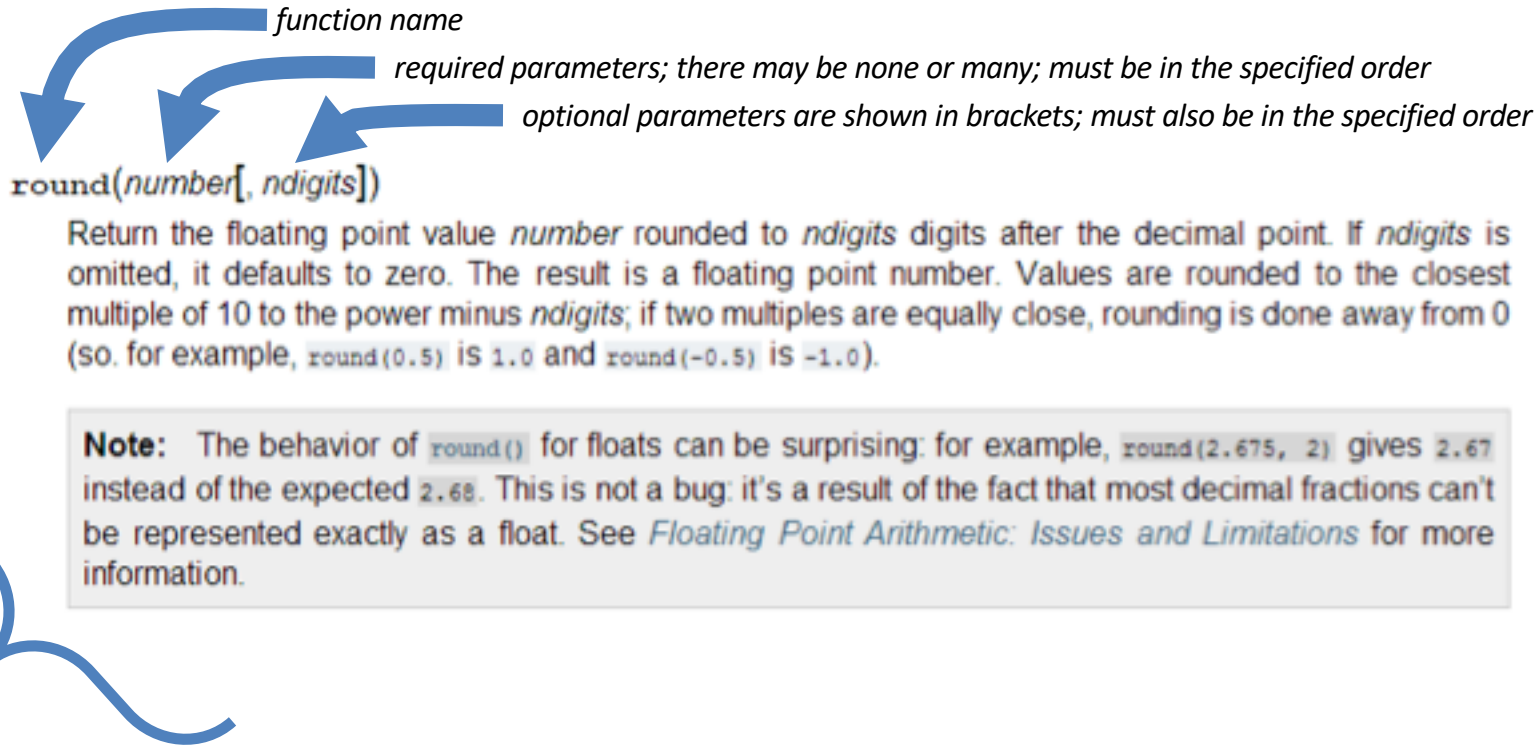
`round`(*number*[, *ndigits*])

Return the floating point value *number* rounded to *ndigits* digits after the decimal point. If *ndigits* is omitted, it defaults to zero. The result is a floating point number. Values are rounded to the closest multiple of 10 to the power minus *ndigits*; if two multiples are equally close, rounding is done away from 0 (so. for example, `round(0.5)` is `1.0` and `round(-0.5)` is `-1.0`).

Note: The behavior of `round()` for floats can be surprising: for example, `round(2.675, 2)` gives `2.67` instead of the expected `2.68`. This is not a bug: it's a result of the fact that most decimal fractions can't be represented exactly as a float. See *Floating Point Arithmetic: Issues and Limitations* for more information.

A quick primer on reading the PyDocs

Example PyDoc entry:



A quick primer on reading the PyDocs

Another example:

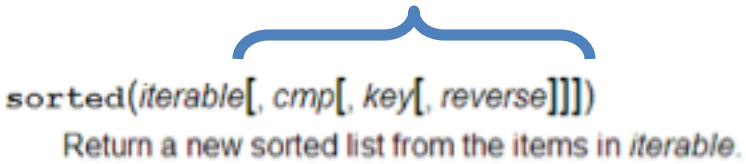
`sorted`(*iterable*[, *cmp*[, *key*[, *reverse*]]])

Return a new sorted list from the items in *iterable*.

A quick primer on reading the PyDocs

Another example:

When brackets are nested like this, it means that to use the more nested parameters, you must also specify the previous parameters (i.e. the ones "less" nested)



Examples:

Allowed:

```
sorted(iterable)
sorted(iterable, cmp)
sorted(iterable, cmp, key)
sorted(iterable, cmp, key, reverse)
```

Not allowed:

```
sorted(cmp)
sorted(iterable, key)
sorted(iterable, reverse, cmp, key)
```

However, you can specify in any order if you refer to parameters by name:

```
sorted(iterable = someIterableThing, reverse = True)
```

5. Commenting your code

Commenting your code

Comments are text you add to your code that is ignored by Python. Comments are meant to help others (and yourself) better understand what your code is doing.

```
# this is a comment
# comments are ignored by Python
print("Hello!”) # you can put them almost anywhere
print("How are you?”) # use them often!
```

What this code prints:

```
Hello!
How are you?
```

Important to note:

If you add a comment in an indented block, the comment must be indented as well (otherwise you will get an error).

Multi-line comments

Single line comments are made using the # sign.

For a multi-line comment, use """ or ''' like so:

Example:

```
"""
This here is a multi-line comment.
Make sure to end it with matching quotes!
"""
print("hello”)
'''
This is another mutli-line comment!
What fun!
'''
```

When should I comment?

- Comments are meant to improve the understandability of your code to another person (and possibly yourself in the future).
- Use them whenever you think a piece of code might be particularly confusing to a reader.
- You can also use them to "section" your code. Sometimes I write comments first, before the code, and use it as an outline for the overall code structure.
- **Most importantly, though**: always keep your comments up to date! Inaccurate comments are worse than no comments at all, because they mislead the reader and can cause false assumptions.

Appendix:
More if/else examples & practice

More practice

What will this code print?

```
yourAge = 50
catsYouOwn = 5
if (catsYouOwn > (yourAge / 10)):
    print("You are officially a cat lady!”)
elif (catsYouOwn == (yourAge / 10)):
    print("Careful! You are close to becoming a cat lady”)
else:
    print("Congrats, you are not a cat lady”)
```

More practice

What will this code print?

```
yourAge = 50
catsYouOwn = 5
if (catsYouOwn > (yourAge / 10)):
    print("You are officially a cat lady!”)
elif (catsYouOwn == (yourAge / 10)):
    print("Careful! You are close to becoming a cat lady”)
else:
    print("Congrats, you are not a cat lady”)
```

Result

```
Careful! You are close to becoming a cat lady
```

More practice

What will this code print?

```
alive = True
breathing = False

if alive and breathing:
    print("Everything is ok!”)
elif alive and not breathing:
    print("You! Go get help!”)
elif not alive and breathing:
    print("Zombie attack?”)
elif not alive and not breathing:
    print (":(”)
```

More practice

What will this code print?

```
alive = True
breathing = False

if alive and breathing:
    print("Everything is ok!”)
elif alive and not breathing:
    print("You! Go get help!”)
elif not alive and breathing:
    print("Zombie attack?”)
elif not alive and not breathing:
    print (":(”)
```

Result

```
You! Go get help!
```

More practice

What will this code print?

```
alive = True
breathing = False

if alive and breathing:
    print "Everything is ok!"
if alive and not breathing:
    print "You! Go get help!"
if not alive and breathing:
    print "Zombie attack?"
if not alive and not breathing:
    print ":("
```

More practice

What will this code print?

```
alive = True
breathing = False

if alive and breathing:
    print "Everything is ok!"
if alive and not breathing:
    print "You! Go get help!"
if not alive and breathing:
    print "Zombie attack?"
if not alive and not breathing:
    print ":("
```

Result

```
You! Go get help!
```

More practice

What will this code print?

```
alive = True
breathing = False

if alive and breathing:
    print "Everything is ok!"
if alive and not breathing:
    print "You! Go get help!"
if not alive and breathing:
    print "Zombie attack?"
if not alive and not breathing:
    print ":("
```

Result

```
You! Go get help!
```

What's the difference?

Here we used only `if` statements instead of `elif`.

This example gives the same result as the previous example, but notice how that might not always be the case!

The setup shown here allows for the possibility that more than one of these statements can be executed, while the previous setup does not.

This could be good, or not, depending on what you want to do. Always think carefully about which approach makes more sense for what you want to accomplish.

More practice

What will this code print?

```
codon = "ATG"
if (len(codon) != 3):
    print "Error, codons must be 3 characters"
else:
    if (codon == "ATG"):
            print "This is a start codon"
    elif (codon == "TAG") or (codon == "TGA") or (codon == "TAA"):
            print "This is a stop codon"
    else:
            print "This is not a start or stop codon"
print "Goodbye!"
```

More practice

What will this code print?

```
codon = "ATG"
if (len(codon) != 3):
    print "Error, codons must be 3 characters"
else:
    if (codon == "ATG"):
            print "This is a start codon"
    elif (codon == "TAG") or (codon == "TGA") or (codon == "TAA"):
            print "This is a stop codon"
    else:
            print "This is not a start or stop codon"
print "Goodbye!"
```

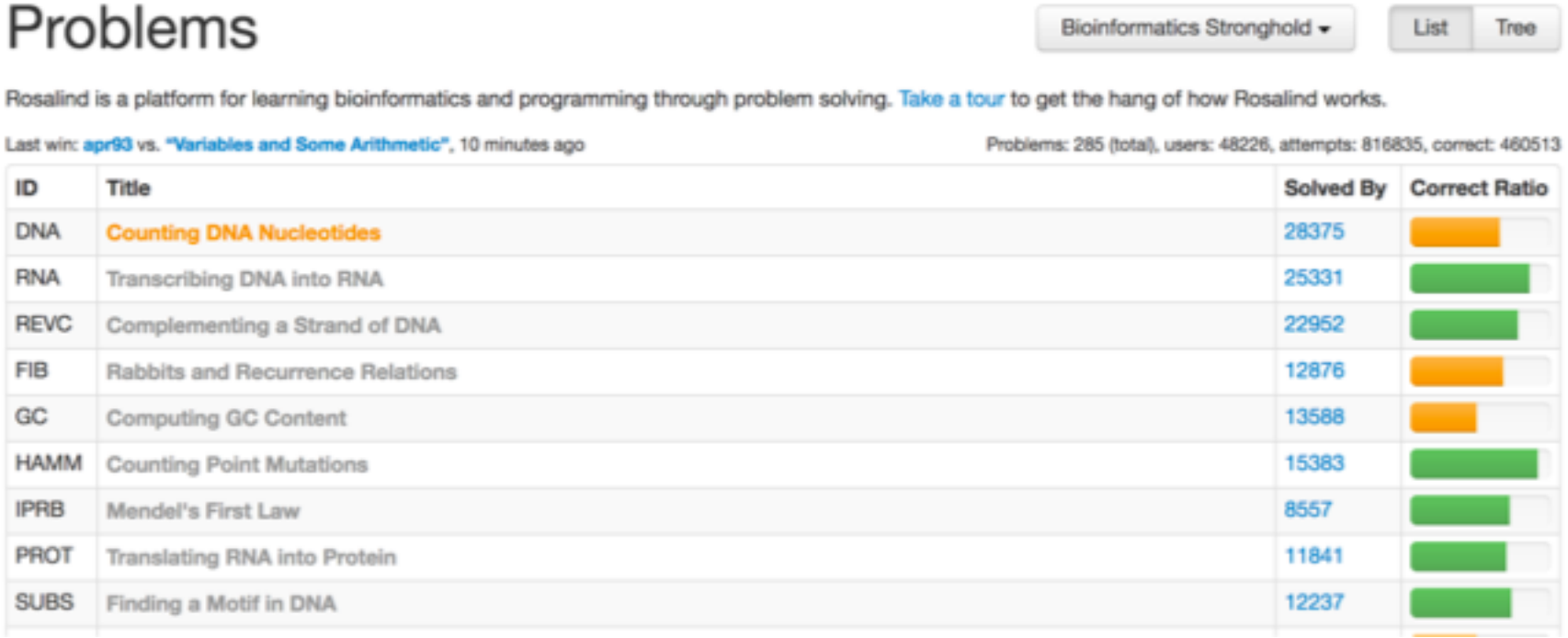
Result

```
This is a start codon
Goodbye!
```

Tips about Programming

- Practice a problem everyday.
- Think of a data task in your lab that you can speed up by using code and program it.
- Rosalind:
  - Practice programming specifically for bioinformatics
  - Unlock new levels and earn badges (gaming!)
- http://rosalind.info/problems/locations/

Rosalind Example



Problems

Bioinformatics Stronghold ▾ | List | Tree

Rosalind is a platform for learning bioinformatics and programming through problem solving. Take a tour to get the hang of how Rosalind works.

Last win: apr93 vs. "Variables and Some Arithmetic", 10 minutes ago

Problems: 285 (total), users: 48226, attempts: 816835, correct: 460513

ID	Title	Solved By	Correct Ratio
DNA	Counting DNA Nucleotides	28375	
RNA	Transcribing DNA into RNA	25331	
REVC	Complementing a Strand of DNA	22952	
FIB	Rabbits and Recurrence Relations	12876	
GC	Computing GC Content	13588	
HAMM	Counting Point Mutations	15383	
IPRB	Mendel's First Law	8557	
PROT	Translating RNA into Protein	11841	
SUBS	Finding a Motif in DNA	12237	

http://rosalind.info/problems/list-view/