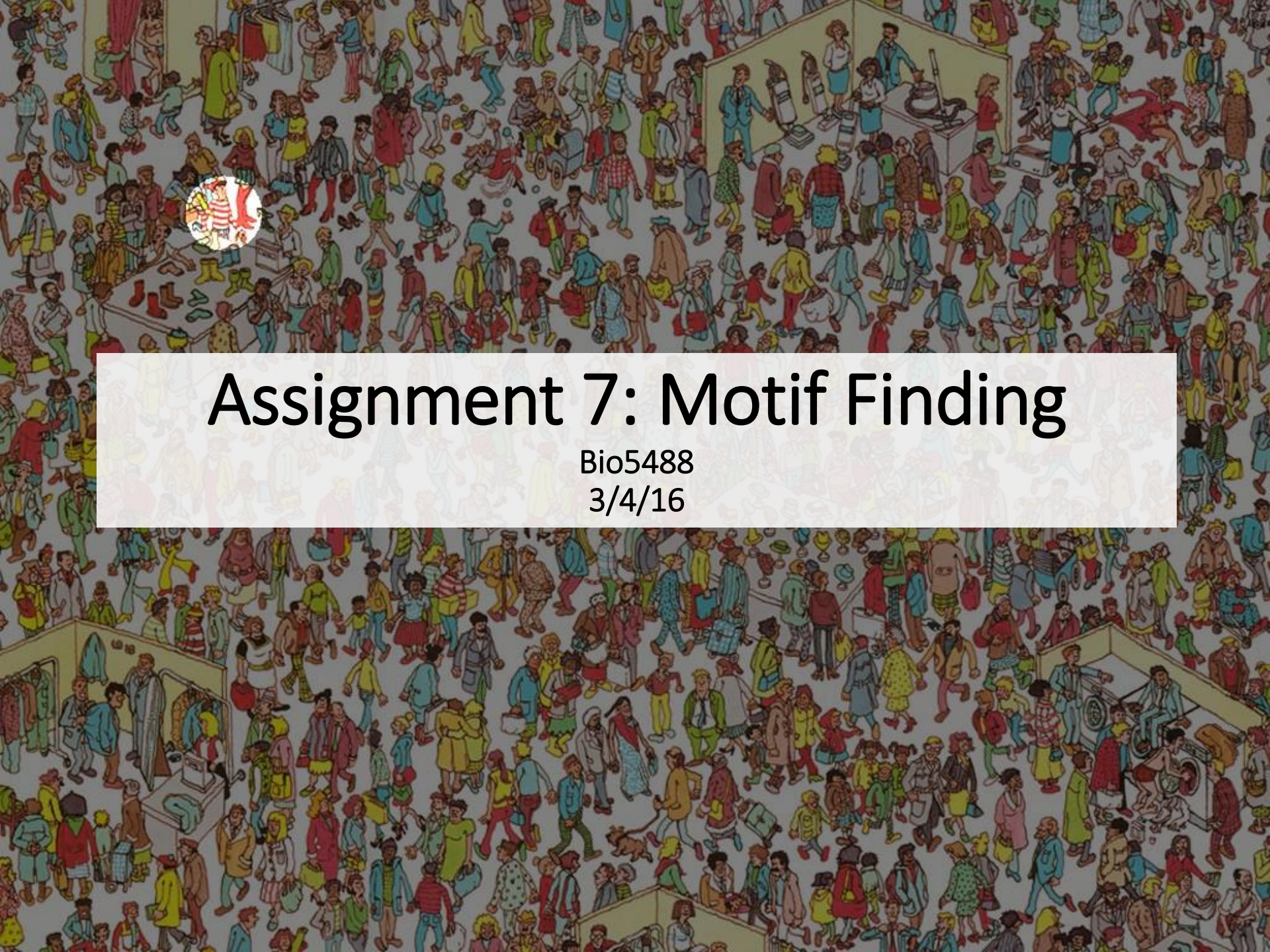


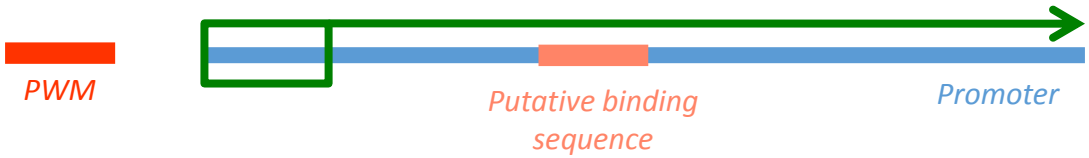
Assignment 7: Motif Finding

Bio5488

3/4/16

Assignment 7: Motif finding

- Input
  - Promoter sequences
  - PWMs of DNA-binding proteins
- Goal
  - Find putative binding sites in the sequences by scanning the sequences for matches to the PWM



- Output
  - List of the locations and scores of putative binding sites

Modified from Jim Havranek's slides

Input files

- Promoter sequences
  - Just the sequence, i.e., not a fasta
- PWMs of DNA-binding proteins
  - *Whitespace*-delimited
  - a_{ij} = score for base *i* at position *j*
    - Rows correspond to A, C, G, & T
    - Columns correspond to positions
    - The higher the score, the better the score

Example PWM

Position

Base	0	1	2	3	4	5
A	-5	-2	9	7	-6	10
C	-3	6	10	2	-5	-2
G	3	-6	-9	-1	5	-10
T	7	-6	-7	-3	10	-4

Example PWM file

```
-5 -9 4 5 -3 2
6 -5 10 -1 0 10
-10 -1 4 3 10 -4
6 0 -1 10 -3 1
```

Assignment TODOs

- Determine the highest affinity binding site for each PWM
  - Calculate by hand or write a script ☺
- Comment the starter script `scan_sequence.py`
  - Comment the existing code blocks
  - Comment the user-defined functions with **function docstrings**

Function docstrings

- Purpose: tells the reader how to use the function
- Guidelines for what to include
  - Describe what the function does
  - Describe the input argument(s)
  - Describe the output value(s)
- Where to learn more:
  - PEP 257: https://www.python.org/dev/peps/pep-0257/
  - Google's Python style guide: http://google-styleguide.googlecode.com/svn/trunk/pyguide.html?showone=Comments#Comments

Example of a function docstring

Summary line

Description of arguments

Description of return value

```
def calc_fishers_linear_discriminant(group1_values, group2_values):
    """Calculates the fisher's linear descriminant between two groups.

    The formula used to calculate fisher's linear discriminant is
    (mean1 - mean2)^2 / (stdev1^2 + stdev2^2)

    Args:
        group1_values: list of floats for group 1
        group2_values: list of floats for group 2

    Returns:
        fisher's linear discriminant

    """

    # Rest of code goes here
```

Retrieving a function's docstring

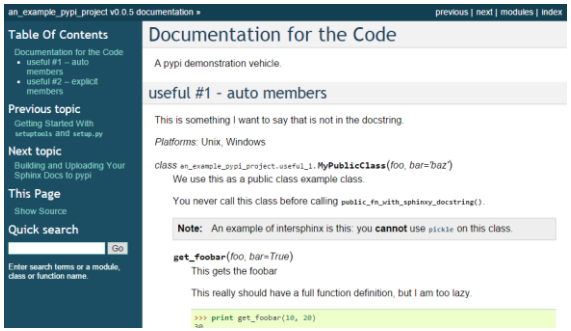
Call `help`

```
>>> help(calc_fishers_linear_discriminant)
```

Function's docstring is returned

```
Help on function calc_fishers_linear_discriminant in module __main__:

calc_fishers_linear_discriminant(group1_values, group2_values)
    Calculates the fisher's linear discriminant between two groups.

    The formula used to calculate fisher's linear discriminant is
    (mean1 - mean2)^2 / (stdev1^2 + stdev2^2)

    Args:
        group1_values: list of floats for group 1
        group2_values: list of floats for group 2

    Returns:
        fisher's linear discriminant
```

Docstrings are also used by third-party programs to create user-friendly documentation for your project



Assignment TODOs (cont.)

- Determine the highest affinity binding site for each PWM
  - Calculate by hand or write a script ☺
- Comment the existing code
  - Comment the user-defined functions with **function docstrings**
- Modify the script to scans the reverse complement of the input sequence
- Modify the script to report only report hits that have scores above a given threshold
- Scan promoters (n = 2) to find putative binding sites for each DNA-binding protein (n = 2)
- Answer follow-up questions

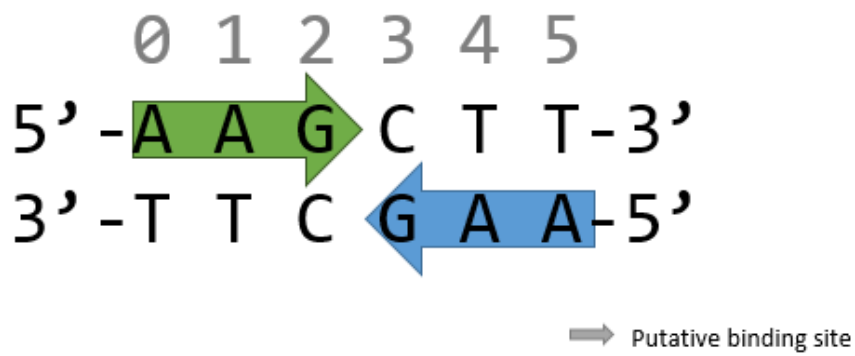
Indexing

- Indexing is somewhat arbitrary; however it's important to follow conventions:
  - The start position of a feature is smaller than the stop position
  - The coordinates are relative to the forward strand

Consensus sequence

A A G

Schematic of putative binding sites



Output table

strand	sequence	position	score
forward	AAG	0	12.8
reverse	AAG	3	15.6

Python list comprehensions

- Purpose: create lists in 1 line of code
  - There are also **dictionary comprehensions** that work similarly

	Code template	Example

As a for loop	`for <item> in <list>:`
	`  <expression>`
		`x = []`
		`for i in range(5):`
		`  x.append(i**2)`

List compre-hension	`[<expression> for <item> in <list>]`	`x = [i**2 for i in range(5)]`

Python list comprehensions *with filtering*

	Code template	Example

As a for loop	`for <item> in <list>:`
	`  if <conditional>:`
	`  <expression>`
		`x = []`
		`for i in range(5):`
		`  if i % 2 == 0: # if i is even`
		`  x.append(i**2)`

List compre-hension	`[<expression> for <item> in <list>` `  if <conditional>]`	`x = [i**2 for i in range(5)` `  if i % 2 == 0]`

- Where to learn more:
  - List comprehension PEP: https://www.python.org/dev/peps/pep-0202/
  - Dict comprehension PEP: https://www.python.org/dev/peps/pep-0274/

Python's `zip` function

- Purpose: "zip" together lists
  - Returns a list* of tuples where the ith tuple contains the ith element from each of the input lists

	Code template	Example

As a for loop	`<zipped_list> = list(zip(<list1>, <list1>, ...))`	`x = [0, 1, 2]`
		`y = [0, 1, 4]`
		`coords = list(zip(x,y))`
		`>>> coords`
		`[(0, 0), (1, 1), (2, 4)]`

- Zipped lists can be unzipped (`zip(*coords)`)
- Where to learn more
  - Python.org documentation: https://docs.python.org/3.4/library/functions.html#zip

**It's really an iterator, one of list's close cousins*

Printing formatted strings in Python with `format`

- Purpose: make your print statements print “pretty” output, e.g., tables
- `format` transforms a “template string” by substituting placeholders with formatted values
  - Placeholders are enclosed in {} and specify how the value should be formatted

Not so pretty	Pretty

`>>> score = 1/300`
`>>> print("The score was " + str(score))`
`The score was 0.0033333333333333335` 
	`>>> print("The score was {s:.3f}".format(s=score))`
	`The score was 0.003` 

- Where to learn more:
  - Python.org tutorial: https://docs.python.org/3.4/tutorial/inputoutput.html#fancier-output-formatting
  - Python.org documentation: https://docs.python.org/3.4/library/string.html#formatstrings
  - Python Course tutorial: http://www.python-course.eu/python3_formatted_output.php

Assignment 7: requirements

- Due in 3 weeks (3/23/15) at 10 AM
- Your submission directory should contain
  - A modified `scan_sequence.py` that is well commented and contains a docstring for each user-defined function
  - A `README.txt` with the answers to the questions and the commands/work you used to arrive at the answer