1 Introduction

This document lists the broad programming topics you should know about for the midterm. It also lists, in cheat-sheet form, the Python syntax and functions you might need to understand or use on the midterm.

2 Programming topics

2.1 General Python stuff

1. Basic expressions: Strings, numbers, booleans (True/False values)
2. Basic arithmetic on numbers; basic logical operators on booleans
3. Assigning names to values with =
4. Calling functions with ()
5. Accessing things: indexing lists and tables with []; accessing attributes and methods with .
6. Assigning other things with =: assigning slots in lists with []; adding columns to tables with []
7. Running code conditionally with if statements
8. Running code iteratively with for loops
9. Defining functions with def statements
10. Returning values from functions
11. Encapsulating blocks of code (or single “ideas”) into functions
12. Thinking about functions as values that happen to be callable with (), and passing functions as arguments to other functions (“higher-order” functions)

2.2 Lists, arrays, and tables

1. Making lists with []
2. Making arrays from lists with np.array(...); making arrays of consecutive numbers with np.arange(...)
3. Making tables by reading data files with Table.read_table(...); directly using the Table(...) function
4. Zero-based indexing for lists and arrays, and the left-inclusive / right-exclusive behavior of np.arange and slice indexing
5. Producing a concatenated list from two lists with +
6. Differences between arrays and lists
7. Basic functions that do things with arrays, like np.sum, np.mean, and np.diff; operators like +, -, *, /, **, and & acting on two arrays or on an array and a single value
8. Accessing columns of a table, which are just arrays
9. Making a table with a subset of the columns in an existing table with .select
10. Making a table with a subset of the rows in an existing table with .where; using logical operations on columns in combination with .where to filter rows according to logical conditions
11. Making a bar chart from a categorical-valued table with .barh
12. Making a histogram from a table with .hist; making a density histogram; controlling the bin widths
13. Applying a function to each element of a column in a table with .apply (a higher-order function)
14. Joining two tables with `.join`

15. Grouping rows of a table together with `.group`; aggregating the groups with a function (making `.group` a higher-order function)

16. Creating a “contingency table” or “pivot table” on two categorical columns of a table with `.pivot`; aggregating the contents of each list-valued cell in the resulting table with a function (making `.pivot` a higher-order function)

17. Sampling rows of a table (producing a new table) with `.sample`

18. Repeatedly sampling from a table, computing a statistic, and displaying the empirical distribution in a histogram (to approximate the probability distribution of the statistic under sampling)
3 Python cheat sheet

This cheat sheet is organized by topic, though some examples serve double-duty to conserve space. Rather than give exhaustive documentation, we have created examples that demonstrate behavior that might be hard to remember.

3.1 General Python stuff

"Hello, world!" # A string-valued expression
1 # An integer-valued expression
1.2 # A float-valued expression
True # A boolean-valued expression
3 ** 4 # An expression whose value is 3 to the 4th power
pow(3, 4) # A function call expression, also 3 to the 4th power
17 % 5 # An expression whose value is 2, the remainder when 17 is divided by 5
(17 % 5) == 2 # An expression whose value is True
"3.5" # An expression whose value is a string
float("3.5") # An expression whose value is the number 3.5
x = [1,2,3] # An assignment statement; [1,2,3] is a list expression
len(x) # A function call expression whose value is 3, the length of the list x
len([1,2,3]) # Also a function call expression with value 3
x[pow(2,1)] # An indexing expression with value 3
x[0:2] # An Alice-indexing expression with value [1,2]
(x + [4,5]) # An expression with value [1,2,3,4,5]; adding lists concatenates
x[0] = 4 # An index assignment statement
t = Table([[0,1,4,9], [0,1,8,27]], ['squares', 'cubes']) # Making a table
t['squares'] # An indexing expression with value equal to np.array([0,1,4,9])
t['powers of two'] = [1,2,4,8] # An index assignment statement
# Attribute access expression with value ['squares', 'cubes', 'powers of two']:
t.column_labels
t.num_rows # The number of rows in t
len(t.rows) # Also the number of rows in t; rows is a list of Row objects in t

3.2 Array-specific stuff

small_primes_array = np.array([2,3,5,7,11])
odd_positive_integers_less_than_nine = np.arange(1, 9, 2)
np.array([1,2,3]) + np.array([2,3,4]) # An array equal to np.array([3,5,7])
np.sum(np.array([-2.2,1.0,0.0])) # -1.2
np.mean(np.array([-2.2,1.0,0.0])) # -0.4
np.diff(np.array([-1,3,2,5,5,0])) # An array equal to np.array([4,-1,3,0,-5])
np.array([1,2,3]) ** 2 # An array equal to np.array([1,4,9])
np.array([1,2,3]) >= 2 # An array equal to np.array([True,True,True])
np.count_nonzero(np.array([True, False, True])) # 2, the number of True values

3.3 Table-specific stuff

u = Table.read_table('some_data_file.csv') # A table built from a data file
t['squares'] + t['cubes']# An expression with value np.array([0,2,12,36])
### Multiple Table and Array Operations

- **Expression with np.array**: `t['squares'] > 3` evaluates to `np.array([False, False, True, True])`.
- **Table Selection**: `t.where(np.array([True, False, False, False]))` selects only the first row of `t`.
- **Table Filtering**: `t.where(t['squares'] > 3)` selects the last and second-to-last rows of `t`.
- **Column Selection**: `t.select(['squares'])` selects only the 'squares' column from `t`.
- **Bar Chart**: `v = Table([[4, 11, 2], ['apples', 'oranges', 'kiwis']], ['count', 'fruit'])` creates a bar chart with bar lengths 4, 11, and 2 for apples, oranges, and kiwis, respectively.
- **Square Root Application**: `t.apply(math.sqrt, 'squares')` applies the square root function to the 'squares' column of `t`.

### Table Join and Group Operations

- **Join Example**: `w = Table([['Ann', 'Bob', 'Cathy', 'Dan'], ['apples', 'oranges', 'peaches', 'apples']])` defines a table `w`.
  - The join operation `j = w.join('favorite fruit', v, 'fruit')` joins tables `w` and `v` based on the 'favorite fruit' column, creating a new table `j` with 3 rows: 'Ann', 'Bob', and 'Dan' with 'favorite fruit' 'apples', 'Cathy' with 'apples', and 'Cathy' with 'peaches'.
- **Group Example**: `w.group('favorite fruit', collect=len)` groups values of 'favorite fruit' and counts them, resulting in a table with values 'apples', 'oranges', and 'peaches' and corresponding length values 2, 1, and 1.

### Table Pivot Operation

- **Pivot Example**: `w['favorite color'] = ['red', 'blue', 'red', 'blue']` defines a 'favorite color' column in `w`.
  - The pivot operation `w.pivot('favorite fruit', 'favorite color', 'name')` creates a pivot table summarizing the data, showing how many people have each combination of 'favorite color' and 'favorite fruit'.

### Sampling Operations

- **Sampling Example**: `N = 6` defines a variable `N`.
  - `potential_numbers = np.arange(1, N+1, 1)` creates a list of numbers from 1 to `N`.
  - `sampling_table = Table([potential_numbers], ['nums'])` creates a sampling table.
  - `three_random_rolls = sampling_table.sample(3, with_replacement=True)` samples 3 random numbers.
  - `three_distinct_random_rolls = sampling_table.sample(3, with_replacement=False)` samples 3 distinct random numbers.

### Additional Operations

- **Demonstrating sample()**: Sampling numbers from 1 to `N`.
- **Demonstrating group()**: Applying a function to each value list.
- **Demonstrating pivot()**: Summarizing data with pivoting.