CS 150 - Final “Cheat Sheet”

Input/Output

- Reading input from the user
  - `input(message)`: Displays *message* to the user and returns what the user typed as a string
- Reading from a file

```
file = open(filename, "r")
for line in file:
    # do something with line (a string)
file.close()
```

- Writing to a file

```
file = open(filename, "w")  # write to file (overwrite any existing content), OR:
file = open(filename, "a")  # append to the end of existing contents
file.write(item)  # writes item to file (e.g. string, number) w/o trailing newline
```

- Reading from a URLs (webpages)

```
import urllib.request
web_page = urllib.request.urlopen(some_url)
for line in web_page:
    line = line.decode('ascii', 'ignore')
    # do something with line (a string)
```

- Command-line arguments

```
import sys
```

`sys.argv` is a list containing the command-line arguments (the first element is the program name)

Strings

- The following functions are built-in and answer questions about strings
  - `len(string)`: Returns the number of characters in the string
  - `int(string), float(string)`: Converts a string to an int or float
- String object methods
  - `upper(), lower(), capitalize()`: Returns a new string that is upper or lower-cased, or capitalized
  - `find(some_string)`: Returns the index that *some_string* occurs at in the string or -1 if it does not occur
  - `find(some_string, index)`: Same as above, but starts searching at *index*
  - `replace(old, new)`: Return a copy of the string with all occurrences of *old* substituted with *new*
  - `startswith(prefix)`: Returns True if the string starts with *prefix*, False otherwise
  - `endswith(suffix)`: Returns True if the string ends with *suffix*, False otherwise
  - `strip()`: Returns a copy of the string with leading and trailing whitespace removed
  - `split()`: Return a list of the words in the string using a space as the delimiter
- String operators
  - `string1 + string2`: Returns a new string that is the concatenation of *string1* and *string2*
  - `string * int`: Returns a new string that is *string* repeated *int* times
  - `substr in string`: Returns True if *substr* is a substring of *string*, False otherwise

Lists

- Creating new lists
  - `[]` creates empty list
  - `[object1, object2, ...]` creates list containing objects
  - `list(iterable)` creates a list from any iterable object (e.g., range, set, string)
- The following functions are built-in and answer questions about lists
  - `len(list)`: Returns the number of elements in *list*
  - `sum(list), min(list), max(list)`: Returns the sum, min, or max of elements in *list*
- List object methods
  - `append(x)`: Adds *x* to the end of the list
  - `extend(other_list)`: Adds all elements of *other_list* the end of the list
  - `find(item)`: Returns the index of the first occurrence of *item* in the list or -1 if it does not occur
  - `insert(index, x)`: Insert *x* at *index* in the list
  - `pop()`: Removes the item at the end of the list and returns it
  - `pop(index)`: Removes item at *index* from the list and returns it
  - `reverse()`: Reverses the elements in the list
  - `sort()`: sorts the elements in the list
- List operators
  - `list1 + list2`: Returns a new list that contains the elements of *list1* followed by the elements of *list2*
  - `list * int`: Returns a new list that contains the items in *list* repeated *int* times
  - `item in list`: Returns True if *item* is an element of *list*, False otherwise

Sets

- Creating new sets
  - `set()` creates empty set
  - `{elt1, elt2, ...}` creates a new set with the given elements
  - `set(iterable)` creates a set from any iterable object (e.g., string, list)
- The following functions are built-in and answer questions about sets
  - `len(set)`**:** Returns the number of elements in the set
- Set object methods
  - `add(elt)`**:** Adds *elt* to the set
  - `clear()`**:** Removes all elements from the set
  - `pop()`**:** Removes an arbitrary element from the set and returns it
  - `remove(elt)`**:** Removes *elt* from the set
- Set operators
  - `elt in set`**:** Returns True if *elt* is an element of *set*, False otherwise
  - `set1 <= set2`**:** Returns True if set1 is a subset of set2 (every element of set1 is in set2) , False otherwise
  - `set1 | set2`**:** Returns union of the two sets (new set with elements from both set)
  - `set1 & set2`**:** Returns intersection of the two sets (new set with elements *common* to both sets)
  - `set1 - set2`**:** Returns set difference (new set with elements *set1* not in *set2*)

Dictionaries

- Creating new dictionaries
  - `{}` creates empty dictionary
  - `{key1:value1, key2:value2, ...}` creates a new dictionary with key-value pairs
- The following functions are built-in and answer questions about dictionaries
  - `len(dict)`**:** Returns the number of entries (key-value pairs) in the dictionary
- Dictionary object methods
  - `clear()`**:** Removes all entries from the dictionary
  - `keys()`**:** Returns an iterable object of the keys in the dictionary
  - `values()`**:** Returns an iterable object of the values in the dictionary
- Dictionary operators
  - `item in dict`**:** Returns True if *item* is in the keys of *dict*, False otherwise

Tuples

- Creating new tuples
  - `()` creates empty tuple
  - `(object1, object2, ...)` creates tuple containing objects
- The following functions are built-in and answer questions about tuples
  - `len(tuple)`**:** Returns the number of elements in the tuple
- Tuple operators
  - `item in tuple`**:** Returns True if *item* is contained in *tuple*, False otherwise
  - `tuple1 + tuple2`**:** Returns a new tuple that is the concatenation of *tuple1* and *tuple2*

Modules

- `turtle` module
  - `forward(distance)`**:** Move the turtle forward by the specified *distance*
  - `right(angle) left(angle)`**:** Turn the turtle right/left by *angle*
  - `goto(x, y)`**:** Move turtle to position *x, y*
  - `setheading(angle)`**:** Set the turtles heading to angle
  - `circle(radius)`**:** Draw a circle with specified *radius*; the center is *radius* units left of the turtle
  - `penup()`**:** Pull the pen up – no drawing when moving
  - `pendown()`**:** Put the pen down – drawing when moving
  - `fillcolor(color)`**:** Change the fill color to *color*, where *color* is a string
  - `begin_fill()`**:** Start filling
  - `end_fill()`**:** Fill in the shape drawn since the last call to `begin_fill`
- `random` module
  - `randint(a, b)`**:** Return a random integer *N* such that $a \le N \le b$
  - `uniform(a, b)`**:** Return a random floating point number *N* such that $a \le N \le b$
- `math` module
  - `sqrt(num)`**:** Return the square root of *num*

- **`matplotlib`** module
  - Importing: **`from matplotlib import pyplot`**
  - **`pyplot.plot(x, y)`**: add data in lists *x* and *y* to the plot
  - **`pyplot.show()`**: display the graph
  - **`pyplot.xlabel(string)`**: label the x-axis with *string* (similarly **`pyplot.ylabel`**)
  - **`pyplot.title(string)`**: set *string* as the title of the plot

Matlab

- Assignment: **`x = 3;    % this is a comment`**
- Results of all operations get printed, unless statements end with a semicolon
- Creating matrices
  - **`[10, 12, 14; 20, 30, 4.4]`**: creates 2x3 matrix (2 rows, 3 cols) with given numbers (commas optional)
  - **`1:7`** creates vector [1 2 3 4 5 6 7]
  - **`zeros(m,n)`**: creates *m* x *n* matrix, all zero (if only one parameter, creates square matrix)
  - **`ones(m,n)`**: creates *m* x *n* matrix, all one
  - **`rand(m,n)`**: creates *m* x *n* matrix with random numbers uniformly distributed between 0 and 1
  - **`eye(n)`**: creates *n* x *n* identity matrix (ones along the diagonal, zero elsewhere)
  - **`[a, b; c d]`**: builds new matrix from matrices a, b, c, d
- Matrix operations
  - **`a'`**: transpose (swap rows and columns)
  - **`a+b  a-b  a.*b  a./b`**: element-wise operations
  - **`a*b`**: matrix multiplication
  - **`a>3`**: element-wise comparison (returns new matrix with 1 and 0 entries)
- Matrix indexing and "slicing" (careful: indexing starts at 1)
  - **`a(2, 1)`**: first element in second row of matrix
  - **`a(:, 1:3)`**: selects all rows, first three columns
- Functions
  - **`disp(x)`**: displays *x* (like "print" in Python) – results of operations also get printed if no trailing semicolon
  - **`[nrows, ncols] = size(m)`**: compute size of matrix
  - **`sum(m), max(m), min(m), mean(m)`**: compute column-wise sum, max, min, mean
- Defining functions:

```
% example function in Matlab (needs to be in file 'my_add.m')
function result = my_add(a, b)
result = a+b   % this is how you return a value
```

R

- Assignment: **`x <- 3   # this is a comment`**
- Creating vectors
  - **`c(10, 12, 14, 20)`**: creates vector with given numbers
  - **`1:7`** creates vector with numbers 1 through 7
  - **`runif(n,0,1)`**: creates vector with *n* random numbers uniformly distributed between 0 and 1
- Vector operations
  - **`a+b  a-b  a*b  a/b`**: element-wise operations
  - **`a>3`**: element-wise comparison (returns boolean vector)
- Vector indexing and "slicing" (careful: indexing starts at 1)
  - **`a[1]`**: first element of vector
  - **`a[3:5]`**: selects elements 3, 4, 5
- Functions
  - **`print(x)`**: prints *x*
  - **`length(x)`**: number of elements in vector
  - **`sum(x), max(x), min(x), mean(x)`**: compute sum, max, min, mean of vector
- Defining functions:

```
# example function in R: adds two numbers together
my_add <- function(a, b){
        return(a + b)
}
```