Nicholas Bowman, Sonja Johnson-Yu, Kylie Jue
CS 106AP

Handout #13
August 14, 2019

Exam Reference Sheet

Note: THIS DOCUMENT WILL BE PROVIDED TO YOU AT THE TIME OF THE FINAL. You do not need to print it out yourself, and it **will not count** towards your two pages of notes.

General Python

Built-in functions

`len()`	Returns the length of a collection (e.g. string or list).
`int()/float()/str()`	Converts a Python object to an integer/float/string type.
`range()`	Generates a range of integer values.
`input()`	Presents a string prompt to the user and returns the string that they input.
`print()`	Display output to the text output area (console).
`min()`	Given an iterable, return the minimum element from that iterable. Can take an optional key function parameter.
`max()`	Given an iterable, returns the maximum element from that iterable. Can take an optional key function parameter.
`sorted()`	Given an iterable, returns the sorted iterable. Can take an optional key function parameter.
`sum()`	Given an iterable, returns the sum of its elements.

Keywords used in boolean expressions

`not`	Negates (flips) the boolean value that follows.
`in`	Indicates if an element is part of a collection of objects.
`and`	Returns True if both values are True, False otherwise.
`or`	Returns False if both values are False, True otherwise.

Strings

Remember that string functions are called using the `noun.verb()` convention. For example,

`str.isalpha()` where **`str`** is the string literal or variable storing the string.

Functions that return booleans

`isalpha()/isdigit()`	Returns a boolean indicating if a string is composed of all letters/digits.
`isupper()/islower()`	Returns a boolean indicating if a string is composed of all uppercase/lowercase characters.

Functions that return strings

`upper()/lower()`	Returns a string with all characters uppercase/lowercase.
`strip()`	Returns a string with leading and trailing whitespace removed.

Functions that return ints

`find()`	Returns the index of the first occurrence of a specified character in the string.

Functions that return lists

`split()`	Returns a list containing all substrings separated by the indicated delimiter.

Lists

To create an empty list, you use square brackets **`[]`**.

Remember that list functions are called using the **`noun.verb()`** convention. For example, **`lst.append()`** where **`lst`** is the variable storing the list.

`append()`	Add an element to the end of the list.
`extend()`	Add all elements in the specified list to the end of the target list.
`pop()`	Remove an element from the list and return it.
`insert()`	Insert an element into the specified index of a list.

Slicing

Recall that you can use slicing on both strings and lists (collections). You can get a particular slice of a collection using **`collection[start_index:end_index:step]`**, where the slice starts at **`start_index`** and stops right before **`end_index`** (not inclusive). The step is optional and defaults to a value of 1.

Dictionaries

- To create an empty dictionary, you use curly brackets **`{}`**.
- You can put or re-assign a key-value pair in your dictionary **`d`** using **`d[key] = value`**.
- To check if a particular key exists inside your dictionary **`d`**, you can use **`key in d`**.

Remember that list functions are called using the **`noun.verb()`** convention. For example, **`d.append()`** where **`dict`** is the variable storing the dict.

`keys()`	Returns an iterable over all of the keys in the dictionary.
`values()`	Returns an iterable over all of the values in the dictionary.
`items()`	Returns an iterable over the key, value pairs in the dictionary.

Recall that you can use iterables in for-each loops. To convert them to lists, you use **`list(iterable)`**. For example, you would need to use **`list(d.keys())`** to create a list of all the keys in a dictionary **`d`**.

Images

SimpleImage Code Patterns

```
# create image from filename
image = SimpleImage(filename)
# create blank image
image = SimpleImage.blank(width, height)

# foreach loop
for pixel in image:
    # do something with pixel

# range/y/x loop
for y in range(image.height):
    for x in range(image.width):
        pixel = image.get_pixel(x, y)

# pixel attributes
pixel.red, pixel.green, pixel.blue
pixel.x, pixel.y
```

File reading

Reading lines from a file

```
with open(filename, 'r') as f:
    for line in f:
        # do something with line
```

Campy

`GWindow` has the following properties and functions:

`window.width`	A property for accessing the window's width
`window.height`	A property for accessing the window's height
`window.add()`	Add a specified object to the window. Can optionally specify the x and y coordinates to place the object add
`window.remove()`	Removes a specified object from the window
`window.clear()`	Clears all content from window and returns it to its blank state
`window.get_object_at()`	Gets the object (if any) located at a given location in the window

Create a GWindow by using the following constructor:

```
window = GWindow(width=100, height=100, title='Breakout')
```

`GObject` has the following properties and functions:

`obj.width`	A property for accessing the object's width
`obj.height`	A property for accessing the object's height
`obj.x`	A property for accessing the object's x-coordinate
`obj.y`	A property for accessing the object's x-coordinate
`obj.color`	A property for accessing the object's outline color. A list of colors supported by campy can be found here.

`obj.fill_color`	A property for accessing the object's interior (fill) color. A list of colors supported by campy can be found here. [**`GRect`** and **`GOval`** only]
`obj.filled`	A property for accessing whether or not the object is filled in (either True or False) [**`GRect`** and **`GOval`** only]
`obj.move(dx,dy)`	Moves the object on the screen using the specified displacements

There are multiple different **`GObject`** shapes:

`GRect`	**`rectangle = GRect(width, height, x=0, y=0)`** where **`(x, y)`** is the upper left corner of the rectangle
`GOval`	**`oval = GOval(width, height, x=0, y=0)`** where **`(x, y)`** is the upper left corner of the bounding rectangle around the oval
`GLine`	**`line = GLine(x0, y0, x1, y1)`** where **`(x0, y0)`** and **`(x1, y1)`** are the starting and ending points for the line
`GLabel`	**`text_label = GLabel(label, x=0, y=0)`** where **`label`** is the string contained inside the text label and **`(x, y)`** is the bottom left corner of the label

Campy allows you to detect four types of mouse movements:

`onmouseclicked(fn)`	Occurs when the user **presses and then releases** any button on their mouse
`onmousereleased(fn)`	Occurs when the user **releases** any button on their mouse
`onmousemoved(fn)`	Occurs when the user **moves** the mouse in any direction
`onmousedragged(fn)`	Occurs when the user **both presses and moves** their mouse