Python Summary

The textbook has many useful tables describing various functions and what they do. Here I have tried to bring everything together so that you can have it as an easy reference. While I have included everything that was in the tables in the book, I have rearranged some parts to avoid duplication and added some extra commands where I thought they may be useful.

Contents

1 Common Syntax

If statements

```
if condition:
    statements
```

```
if condition:
    statements
else:
    statements
```

```
if condition1:
    statements
elif condition2:
    statements
elif condition3:
    statements
else:
    statements
```

Loop statements

While statements:

```
while condition:
    statements
```

For statements:

```
for var in seq:
    statements
```

```
for var in range(limit):
    statements
```

Other statements

Function definition

```
def name(parameters):
    statements
```

Return statement

```
return value
```

2 Built-in Functions

Built-in Python Functions

Built-in functions

`abs`(x)	Returns the absolute value of x.
`max`(x, y, ...)	Returns the largest of the arguments.
`min`(x, y, ...)	Returns the smallest of the arguments.
`round`(x)	Returns the closest integer to x.
`int`(x)	Converts x to an integer.
`float`(x)	Converts x to a floating-point number.
`len`(s)	Returns the length of the string argument s.
`str`(x)	Converts x to a string.

3 Common Libraries

Selections from the `math` library

Mathematical constants

`pi`	The mathematical constant π.
`e`	The mathematical constant e (the base for natural logarithm).

General mathematical functions

`sqrt`(x)	Returns the square root of x.
`floor`(x)	Returns the largest integer less than or equal to x.
`ceil`(x)	Returns the smallest integer greater than or equal to x.
`copysign`(x,y)	Returns x with the sign of y.

Logarithmic and exponential functions

`exp`(x)	Returns the exponential function of x (e^x).
`log`(x)	Returns the natural logarithm (base e) of x.
`log10`(x)	Returns the common logarithm (base 10) of x.

Trigonometric functions

`cos`(*theta*)	Returns the cosine of the radian angle *theta*.
`sin`(*theta*)	Returns the sine of the radian angle *theta*.
`tan`(*theta*)	Returns the tangent of the radian angle *theta*.
`atan`(x)	Returns the principal arctangent of x, which lies between $-\pi/2$ and $+\pi/2$.
`atan2`(y,x)	Returns the angle between the x-axis and the line from the origin to (x, y).
`radians`(*angle*)	Converts an angle measured in degrees to radians.
`degrees`(*angle*)	Converts an angle measured in radians to degrees.

Selections from the `random` library

Random integers

`randint(`*min, max*`)`	Returns a random integer between *min* and *max*, inclusive.
`randrange(`*limit*`)`	Returns a random integer from 0 up to but not including *limit*.
`randrange(`*start, limit*`)`	Returns a random integer from *start* up to but not including *limit*.

Random floating-point numbers

`random()`	Returns a random floating-point number in the range between 0 and 1.
`uniform(`*min, max*`)`	Returns a random floating-point number between *min* and *max*.

Random functions on lists

`choice(`*list*`)`	Returns a random element from the specified list.
`sample(`*list, k*`)`	Returns a list with k elements randomly chosen from *list*.
`shuffle(`*list*`)`	Rearranges the list in a random order.

Initialization functions

`seed()`	Randomizes the internal number generator.
`seed(`*k*`)`	Sets the internal state of the random number generator so that it generates the same sequence for any specific value of the integer k.

4 Portable Graphics Library

Methods for `GWindow` objects

`GWindow(`*width, height*`)`	Creates a new `GWindow` objects of the specified size.
gw`.get_width()`	Returns the width of the graphics window.
gw`.get_height()`	Returns the height of the graphics window.
gw`.add(`*obj*`)`	Adds the object to the graphics window.
gw`.add(`*obj, x, y*`)`	Repositions the object to (x, y) and adds it to the window.
gw`.remove(`*obj*`)`	Removes the object from the graphics window.
gw`.clear()`	Removes all objects from the graphics window.
gw`.get_element_at(`*x, y*`)`	Returns the topmost graphical object covering the point (x, y). If no such object exists then `None` is returned.
gw`.add_event_listener(`*type, func*`)`	Prepares the window to respond to events of the specified *type* by calling *func*.
gw`.set_interval(`*func, delay*`)`	Prepares the window to repeatedly call *func* every *delay* milliseconds.
gw`.set_timeout(`*func, delay*`)`	Prepares the window to call *func* once after waiting *delay* milliseconds.

Creating Graphical Objects

Creating graphical objects

`GRect(`*x, y, width, height*`)`	Creates a `GRect` object with the specified dimensions.
`GRect(`*width, height*`)`	Creates a `GRect` object an $(0, 0)$ with the specified size.
`GOval(`*x, y, width, height*`)`	Creates a `GOval` object that fits inside the corresponding rectangle of the specified size.
`GOval(`*width, height*`)`	Creates a `GOval` object in which the oval ifts inside a rectangle of the specified size. The origin of the `GOval` is at $(0, 0)$.
`GLine(`x_1, y_1, x_2, y_2`)`	Creates a `GLine` object connection (x_1, y_1) and (x_2, y_2).
`GLabel(`*str, x, y*`)`	Creates a `GLabel` object containing the specified string with its baseline origin at the point (x, y).
`GLabel(`*str*`)`	Creates a `GLabel` object containing the specified string with its baseline origin at the point $(0, 0)$.
`GArc(`*x, y, width, height, start, sweep*`)`	Creates a `GArc` object at the specified point and dimensions which starts at *start* degrees and extends counterclockwise *sweep* degrees.
`GPolygon()`	Creates an empty `GPolygon` object, which then needs vertices to be added.
`GCompound()`	Creates an empty `GCompound` object, into which other objects can then be added.

Methods available to all objects

Methods that control the location

object`.set_location(`*x, y*`)`	Sets the location of this object to (x, y).
object`.move(`*dx, dy*`)`	Moves the object using the displacements dx and dy.
object`.move_polar(`*r, theta*`)`	Moves the object r pixels in the direction *theta*.

Methods that control the appearance

object`.set_color(`*color*`)`	Sets the color used to display this object.
object`.set_line_width(`*width*`)`	Sets the width of the lines (in pixels) used to draw the object.
object`.set_visible(`*flag*`)`	Sets whether the object is visible, where $flag$ is a boolean.
object`.rotate(`*theta*`)`	Rotates the object *theta* degrees about its origin.
object`.scale(`*sf*`)`	Scales the object by sf both horizontally and vertically.

Methods that control the stacking order

object`.send_backward()`	Moves this object one step backward in the stacking order.
object`.send_forward()`	Moves this object one step forward in the stacking order.
object`.send_to_back()`	Moves this object to the back of the stacking order.
object`.send_to_front()`	Moves this object to the front of the stacking order.

Methods that return properties

object`.get_x()`	Returns the x coordinate of the object.
object`.get_y()`	Returns the y coordinate of the object.
object`.get_width()`	Returns the width of the object.
object`.get_height()`	Returns the height of the object.
object`.get_color()`	Returns the color used to display this object.
object`.get_line_width()`	Returns the width of the lines used to draw the object.
object`.is_visible()`	Returns a boolean indicating whether the object is currently visible.
object`.contains(`*x, y*`)`	Check to see whether the point (x, y) is inside the object.

Methods only available to `GFillableObject` objects

`GFillableObjects` include `GRects`, `GOvals`, `GArcs`, and `GPolygons`

object`.set_filled(`*bool*`)`	Sets whether the object is filled.
object`.set_fill_color(`*color*`)`	Sets the color used to fill the interior of the object.
object`.get_fill_color()`	Returns the color used to display the interior of the object.
object`.is_filled()`	Returns a boolean indicating whether the object is currently filled.

Methods only available to `GLabel` objects

object`.set_font(`*str*`)`	Sets the font for the label. The format of the font specification is a CSS string as described in the text.
object`.set_label(`*str*`)`	Sets the text of the label to the provided string.
object`.get_label()`	Returns the text of the label as a string.
object`.get_ascent()`	Returns the *font ascent* (the maximum distance above the baseline).
object`.get_descent()`	Returns the *font descent* (the maximum distance below the baseline).

Methods only available to `GLine` objects

object`.set_start_point(`*x, y*`)`	Changes the starting point of the line to (x, y) without changing the end.
object`.set_end_point(`*x, y*`)`	Changes the end point of the line to (x, y) without changing the start.
object`.get_start_point()`	Returns the starting point of the line.
object`.get_end_point()`	Returns the end point of the line.

Methods only available to `GRect` or `GOval` objects

object`.set_size(`*width, height*`)`	Sets the size of the object to the specified width and height.
object`.set_bounds(`*x, y, width, height*`)`	Sets both the location of the object and the size of the object.

Methods only available to `GArc` objects

arc`.set_start_angle(`*start*`)`	Sets the start angle to *start* degrees.
arc`.get_start_angle()`	Returns the start angle.
arc`.set_sweep_angle(`*sweep*`)`	Sets the sweep angle to *sweep*.
arc`.get_sweep_angle()`	Returns the sweep angle.
arc`.get_start_point()`	Returns the coordinate of the starting point of the arc.
arc`.get_end_point()`	Returns the coordinate of the ending point of the arc.

Methods only available to `GPolygon` objects

poly`.add_vertex(`*x, y*`)`	Adds a vertex at the point (x, y).
poly`.add_edge(`*dx, dy*`)`	Adds a vertex shifted by dx and dy from the preceeding vertex.
poly`.add_polar_edge(`*r, theta*`)`	Adds a vertex shifted by r units in direction *theta*.
poly`.get_bounds()`	Returns a `GRect` object of the bounding rectangle of the polygon.

Methods in the `GImage` class

Creating `GImages`

`GImage(`*filename*`)`	Creates a new `GImage` by reading the image data from the specified file.
`GImage(`*array*`)`	Creates a new `GImage` from the provided pixel array.

Method to read pixels

image`.get_pixel_array()`	Returns the pixel array for this image.

Static methods relating to `GImages`

`GImage.get_red(`*pixel*`)`	Returns the red component of the pixel as an integer between 0 and 255.
`GImage.get_green(`*pixel*`)`	Returns the green component of the pixel as an integer between 0 and 255.
`GImage.get_blue(`*pixel*`)`	Returns the blue component of the pixel as an integer between 0 and 255.
`GImage.create_rgb_pixel(`r, g, b`)`	Creates a pixel value from the specified r, g, and b components, each of which is between 0 and 255.
`GImage.create_rgb_pixel(`a, r, g, b`)`	Creates a pixel value from the specified r, g, and b components and with opacity a, each of which is between 0 and 255.

5 String Methods

Common methods in Python's string class

Finding patterns

str.`find`(*pattern*)	Searches the string *str* for the string *pattern*, starting at the beginning of *str*. Returns the first index at which the pattern appears, or -1 if not found.
str.`find`(*pattern, k*)	Same as above, but starts the search at index k.
str.`rfind`(*pattern*)	Searches backward in *str* for the last instance of *pattern*, starting at the end of *str*. Returns the last index at which the pattern appears, or -1 if not found.
str.`rfind`(*pattern, k*)	Same as above, but starts at index k.
str.`startswith`(*prefix*)	Returns `True` if *str* starts with the characters in $prefix$.
str.`endswith`(*suffix*)	Returns `True` if *str* ends with the characters in $suffix$.

Creating transformed strings

str.`lower()`	Returns a copy of *str* will all letters converted to lowercase.
str.`upper()`	Returns a copy of *str* with all letters converted to uppercase.
str.`capitalize()`	Returns a lowercase copy of *str* but with the first letter capitalized.
str.`lstrip()`	Returns a copy of *str* after removing any whitespace characters from the left side.
str.`rstrip()`	Returns a copy of *str* after removing any whitespace characters from the right side.
str.`strip()`	Returns a copy of *str* after removing any whitespace characters from both sides.
str.`replace`(*old, new*)	Returns a copy of *str* after replacing all instances of the string *old* with the string *new*.

Testing for character properties

str.`isalpha()`	Returns `True` if *str* is nonempty and contains only letters.
str.`isdigit()`	Returns `True` if *str* is nonempty and contains only numeric digits.
str.`isalnum()`	Returns `True` if *str* is nonempty and contains only letters or digits.
str.`islower()`	Returns `True` if *str* has at least one letter and all letters are lowercase.
str.`isupper()`	Returns `True` if *str* has at least one letter and all letters are uppercase.
str.`isspace()`	Returns `True` if *str* is nonempty and contains only whitespace characters (eg. space, tab, or newline).

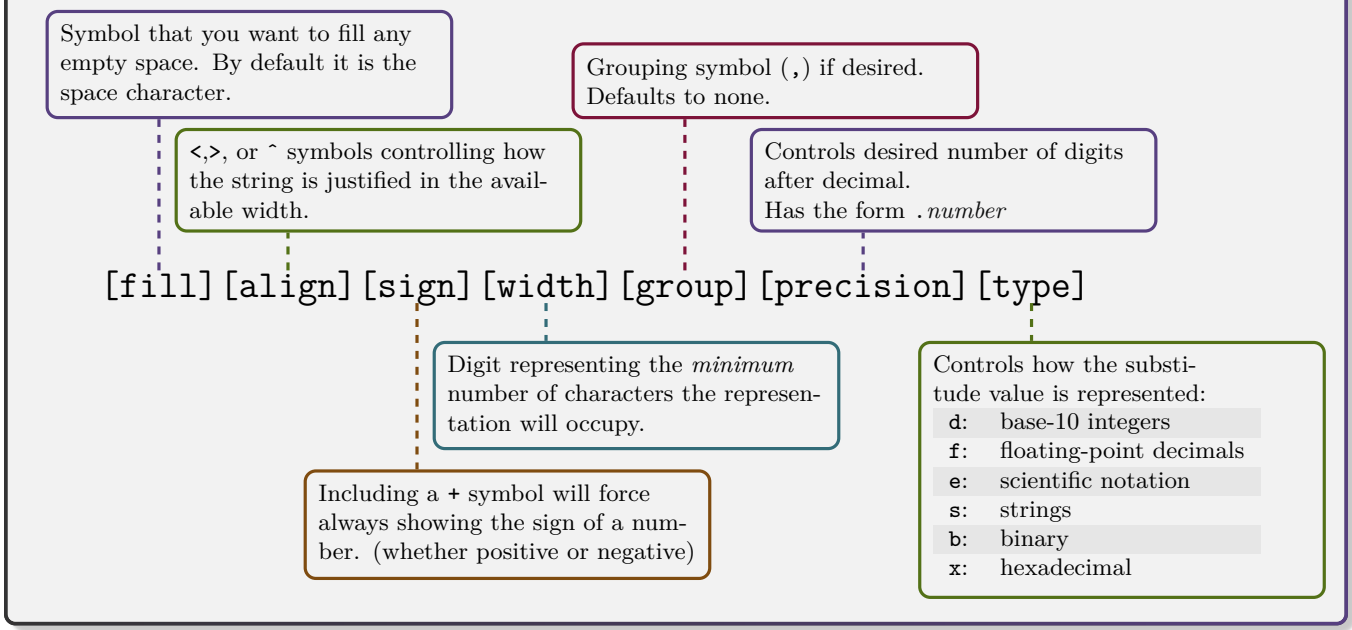
Formatting strings

str.`center`(*width*)	Returns a copy of *str* centered in a field of the specified *width*.
str.`ljust`(*width*)	Returns a copy of *str* flush to the left in a field of the specified *width*.
str.`rjust`(*width*)	Returns a copy of *str* flush to the right in a field of the specified *width*.

Splitting and joining strings

str.`split`(*pattern*)	Splits the *str* into a list of strings by dividing it at *pattern*.
str.`splitlines()`	Splits a multiline string into a list of the individual lines.
sep.`join`(*list*)	Joins the elements of —list— into a string using *sep* to separate the elements.

Format Spec components



6 List Methods

Common methods in Python's list class

Methods that leave the original list unchanged

list.`index(`*value*`)`	Returns the first index matching *value*. This method raises a `ValueError` exception if no match is found.
list.`index(`*value, start*`)`	Starting from *start*, returns the first index matching *value*. This method raises a `ValueError` exception if no match is found.
list.`count(`*value*`)`	Returns the number of times that *value* appears in the list.
list.`copy()`	Returns a *shallow* copy of the original list.

Methods that add or remove elements

list.`append(`*value*`)`	Adds *value* to the end of the list.
list.`extend(`$list_2$`)`	Adds the elements in $list_2$ to the end of the list.
list.`insert(`*index, value*`)`	Inserts *value* before the specified *index* position.
list.`remove(`*value*`)`	Removes the first instance of *value* from the list, raising a `ValueError` exception if it does not appear.
list.`pop()`	Removes and returns the last element of the list.
list.`pop(`*index*`)`	Removes and returns the element at the specified *index* position.
list.`clear()`	Removes all elements from a list.

Methods that reorder the elements of a list

list.`reverse()`	Reverses the order of the elements in the list.
list.`sort()`	Sort the elements of the list in ascending order.
list.`sort(`*key*`)`	Sort the elements of the list in ascending order according to a particular *key*.
list.`sort(`*key*`, reverse=True)`	Sort the elements of the list in descending order according to a particular *key*.

String methods that involve lists

str.`split()`	Splits the string *str* into a list of substrings by dividing it at each occurrence of a whitespace character.
str.`split(`*separator*`)`	Splits the string *str* into a list of substrings by dividing it at each instance of the string *separator*.
str.`splitlines()`	Splits *str* into a list of substrings by dividing it at each line break.
sep.`join(`*list*`)`	Joins the elements of *list* into a string using *sep* to separate the elements.

7 Dictionary Methods

Common methods in Python's dict class

`len(`*dict*`)`	Returns the number of key-value pairs in the dictionary.
dict`.clear()`	Removes all key-value pairs from the dictionary, leaving it empty.
dict`.copy()`	Creates and returns a shallow copy of this dictionary.
dict`.get(`*key, value*`)`	Returns the value associated with *key* in the dictionary, or the specified *value* if *key* is not found. The default value for *value* is `None`.
dict`.setdefault(`*key, value*`)`	If *key* is not in the dictionary, `setdefault` creates a new entry for the key-value pair. The method then returns the value, which is either its previous value or the newly created one.
dict`.pop(`*key*`)`	Removes and returns the key-value pair corresponding to *key*. If the key is not found, then a `KeyError` exception is raised.
dict`.items()`	Returns an iterable object that returns successive tuples consisting of a key-value pair.

8 Set Methods

Common operators and methods in Python's set class

`set()`	Creates an empty set. The expression `{ }` creates an empty dictionary.
`set(`*iterable*`)`	Creates a set from the elements of the iterable object. (Which could be a list, string, etc. Anything you could loop over.)
`len(`*set*`)`	Returns the number of elements in the set.
set_1`.union(`set_2`)` set_1 `|` set_2	Returns the union of set_1 and set_2.
set_1`.intersection(`set_2`)` set_1 `&` set_2	Returns the intersection of set_1 and set_2.
set_1`.difference(`set_2`)` set_1 `-` set_2	Returns the difference of set_1 and set_2.
set_1`.symmetric_difference(`set_2`)` set_1 `^` set_2	Returns the symmetric difference of set_1 and set_2.
set_1 `==` set_2	Returns `True` if set_1 and set_2 contain the same elements.
set_1 `<=` set_2	Returns `True` if set_1 is a subset of set_2.
set_1 `<` set_2	Returns `True` if set_1 is a proper subset of set_2.
element `in` *set*	Returns `True` if *element* is in the *set*.
set_1`.isdisjoint(`set_2`)`	Returns `True` if set_1 and set_2 contain no elements in common.
set`.clear()`	Removes all elements from the set, leaving it empty.
set`.copy()`	Creates and returns a shallow copy of the set.
set`.add(`*element*`)`	Adds the specified element to the set.
set`.remove(`*element*`)`	Removes the element from the set, raising a `ValueError` if it is missing.
set`.discard(`*element*`)`	Removes the element from the set, doing nothing if it is missing.