**while**

* indefinite iteration
* include some kind of incremental counter aNumber = aNumber + 1
* while aNumber <= aBound:
* to **validate** input; when you want to make sure the user has entered valid input for a prompt

**for**

* definite iteration based on list created by range
* if you know max # of execution needed
* for aNumber in range(1, aBound + 1):

**Escape sequence**

* \t (new tab)
* Because of the tab characters between the columns, the position of the second column does not depend on the number of digits in the first column.
* \n (newline)

**Image Processing**

| **Method Name** | **Example** | **Explanation** |
| --- | --- | --- |
| Pixel(r,g,b) | Pixel(20,100,50) | Create a new pixel with 20 red, 100 green, and 50 blue. |
| getRed() | r = p.getRed() | Return the red component intensity. |
| getGreen() | r = p.getGreen() | Return the green component intensity. |
| getBlue() | r = p.getBlue() | Return the blue component intensity. |
| setRed() | p.setRed(100) | Set the red component intensity to 100. |
| setGreen() | p.setGreen(45) | Set the green component intensity to 45. |
| setBlue() | p.setBlue(156) | Set the blue component intensity to 156. |

|  |  |  |
| --- | --- | --- |
| Image(filename) | img = image.Image(“cy.png”) | Create an Image object from the file cy.png. |
| EmptyImage() | img = image.EmptyImage(100,200) | Create an Image object that has all “White” pixels |
| getWidth() | w = img.getWidth() | Return the width of the image in pixels. |
| getHeight() | h = img.getHeight() | Return the height of the image in pixels. |
| getPixel(col,row) | p = img.getPixel(35,86) | Return the pixel at column 35, row 86. |
| setPixel(col,row,p) | img.setPixel(100,50,mp) | Set the pixel at column 100, row 50 to be mp. |

**Nested Iteration**

* **outer iteration** and the **inner iteration (move faster)**
* In the RGB colour model, we can consider the opposite of the red component as the difference between the original red and 255. For example, if the original red component was 50, then the opposite, or negative red value would be 255-50 or 205

**Lists**

* Index, elements, multiple data types within a list
* A list within another list: **nested**. inner list is a **sublist (counts as 1 item in the top level).**
  + nested[3][1] to access item in sublist
* Accessing elements: index operator []
  + numbers[2]
  + .count(): count number of times the item appears
* List membership: in and not in (gives a boolean)
* Concatenation (+) & Repetition (\*): creates new lists and not sublists. Concatenate uses accumulator pattern.
* Id tag
  + id(a\_list)
* Slice: splitting/slicing up the list, omit 1st/2nd index to go the extreme. No new list is created
  + a\_list[1:5:2] *# [start:end+1:step]*
  + a\_list[-4:5] *# gives [3, 55, 23, 1] for [4, 3, 55, 23, 1]*
  + a\_list[:4] *# start till 3rd element*
* Modify: change element in the list
  + a\_list[1] = ‘cat’
  + a\_list[1:3] = [4] *# changes item 1 & 2 to [4]*
* Mutable: use index operator, slice operator and assignment (if -index, counts from the back)
  + Can remove elements by assigning [] or add a list into specified position
  + a\_list = ['a', 'd', 'f']
  + a\_list[1:1] = ['b', 'c'] becomes ['a', ‘b’, ‘c’, 'd', 'f']
* Delete: s
  + del a\_list[1]
  + a\_list.remove(item)
  + a\_list.pop(index) *# deletes and returns last function*
* Extend: adds a list/element onto a list
  + a\_list.extend([11, 24, 55])
  + origlist.append(item)
  + a\_list.insert(3, “cat”)
  + c\_list = a\_list + b\_list
  + [1, ["hello", "goodbye"]]\*2 >> [1, ["hello", "goodbye"], 1, ["hello", "goodbye"]]
* Cloning: creates a new list object
  + b = a [:] *# shallow copy*
* list: tries to turn input into list, not applicable for int (convert to str first)
  + xs = list("Crunchy Frog")
* Some operators like append, sort, reverse returns None
* Aliasing: same object different name
* Is operator: return true if the two references are to the same object.
  + Strings (immutable) of same values are the same objects
  + Lists (mutable) of same values are different objects

| **Method** | **Parameters** | **Result** | **Description** |
| --- | --- | --- | --- |
| append | item | mutator | Adds a new item to the end of a list |
| insert | position, item | mutator | Inserts a new item at the position given |
| pop | none | hybrid | Removes and returns the last item |
| pop | position | hybrid | Removes and returns the item at position |
| sort | none | mutator | Modifies a list to be sorted |
| reverse | none | mutator | Modifies a list to be in reverse order |
| index | item | return idx | Returns the position of first occurrence of item |
| count | item | return ct | Returns the number of occurrences of item |
| remove | item | mutator | Removes the first occurrence of item |

L-systems

* [heading x y]

Modifiers: functions that take lists as arguments and change (side effects) them during execution

* Opposites of pure functions

List comprehension

a\_list = [<expression> **for** <item> **in** <sequence> **if** <condition>]

**Strings**

* .split() uses whitespace (or delimiter in parameter) characters to break string into list of words
* glue.join(<list of separated words>) where glue = ‘<separator string like ;>’

**Tuples**

* immutable, comma-separated sequence of values
* to create new tuple: make new variable, split, add then join parts
  + fruits = tuple(fruitslist)
* assignment: number of variables on the left (tuple of variables) must match the number of elements in the tuple on the right (tuple of values)
* Problem: Input & Output
* Calculate:
* is operator: return true if the two references are to the same object. Since strings are immutable, Python can optimize resources by making two names that refer to the same string literal value refer to the same object.
* **Immutable**: mutable objects can change their state or contents and immutable objects can’t change their state or content (after it is created).
  + Immutable: tuple, string, int, float, bool, keys of dict
  + Mutable: list, dict, set

**Documentation**

Contents >> API overview >> matplotlib.pylot >>