**Python Reference**

**Arithmetic Operators**

|  |  |
| --- | --- |
| **Operator** | **Meaning** |
| + | Addition |
| - | Subtraction |
| \* | Multiplication |
| / | Division- results in float value |
| % | Modulus- gives remainder of floor division |
| // | Floor division- divides, but leaves out the remainder after 0 |
| \*\* | Exponent |

**Comparison Operators**

|  |  |
| --- | --- |
| **Operator** | **Meaning** |
| > | Greater than |
| < | Less than |
| >= | Greater than or equal to |
| <= | Less than or equal to |
| == | Checks if equal |
| != | Checks if not equal |

**Logical Operators**

|  |  |
| --- | --- |
| **Operator** | **Meaning** |
| and | True if both are true |
| or | True if one is true |
| not | True if false |

**Bitwise Operators**

|  |  |
| --- | --- |
| **Operator** | **Meaning** |
| & | Bitwise and (1 if both are 1) |
| | | Bitwise or (1 if 1 is 1) |
| ~ | Bitwise not |
| ^ | Bitwise XOR (1 if 1 is 1, but not both) |
| >> | Bitwise right shift (shifts 1s right) |
| << | Bitwise left shift (shifts 1s left) |

**Special Operators**

|  |  |
| --- | --- |
| **Operator** | **Meaning** |
| is | True if refer to the same object |
| is not | True if do not refer to the same object |
| in | True if x in sequence |
| not in | True if x not in sequence |

**Type examples:**

* Int = 1
* Float = 1.0
* String = “one”
* Bin = 0b1
* List = [1, 2, 3]
* Dictionary = {1 : A, 2 : B, 3 : C}

**Basic built in Functions**

* abs()- returns absolute value
* int()- returns integer value
* str()- returns string value
* float()- returns float value
* bin()- returns binary
* print()- prints value
* range(start, stop, step)- makes list
  + start: inclusive, standard is 0
  + stop: exclusive, standard is length
  + step: how much is goes by, can be positive or negative, default is 1

**Functions**

* def name(parameters):

**Conditions**

* if condition:
  + executes if evaluates true
* elif condition:
  + executes when if statement is false but new condition is true
* else:
  + executes if nothing else is true

**loops: while/else**

* while condition:
  + loops as long as condition is true
* else:
  + executed if break isn’t used in loop

**loops: for/else**

* for i in list:
  + cycles I over list values
  + range() can be used to make a list of indexes
* else:
  + executes if break is not used in loop

**Lists**

* Calling an element by index: a[0]
* Cutting a list: a[start : stop : stride ]
  + start stop stride work like range
* List comprehension: a = [x for x in list if condition]
  + Makes list, cycling over list, outputting x in beginning depending on other conditions
* sorted(list)
  + sorts list
  + makes new list
  + if True is in function, it will go in reverse
* list.sort()
  + sorts list
  + does not make new list
* sum(iterable, [start])
  + iterable is list
  + start is inclusive, tells where to start adding going left to right
  + returns sum
* max(iterable)
  + finds max in list
  + iterable can be replaced with two values for comparison
* min(iterable)
  + finds min in list
  + iterable can be replaced with two values for comparison
* len(list)
  + finds the length of a list
* reversed(list)
  + reverses order of list
* list = [thing] \* number
  + makes list of thing repeated
* list.append(thing)
  + adds thing to end of list
* list.insert(index, thing)
  + adds thing at index
* list.index(thing)
  + returns index at which thing is
* del list[index]
  + deletes thing at index
* list.remove(thing)
  + removes thing from list
* list.pop(index)
  + removes and returns thing at index
* filter(lambda x: condition, list)
  + makes new list, based on condition

**Strings**

* string[index] = character at index in string
* len(string)
  + gives length of string
* string.lower()
  + makes string lowercase
* string.upper()
  + makes string uppercase
* Concatenation 🡪 “string ” + “string” = string string
* “string %s” % (string)
  + Prints as “string string”
* Splitting 🡪 “string” [start : stop]
  + Creates slice
* “/n” – newline
* Print string,
  + Does not put space after print, will print right beside

**Dicitionaries**

* Work like lists
* Have key and value
* Call value by key (as opposed to index)
* Has .items() method
  + .items prints keys and items (no order)

**Classes**

* Class Name(object):

Def \_\_init\_\_(self, inputs)

* Basic class inherits from object class
* Variables outside of functions can be assigned and modified
* \_\_init\_\_ method defines initialization of class
* Different methods can be added using def

**File input/ output**

* File = open(“filename.txt”, “mode”)
* File.write(thing)
  + Writes to file
* File.read()
  + Reads whole file
* File.readline(characters)
  + Reads line of file, subsequent calls go to next line
* File.close()
  + Closes file, must be done to get stuff out of buffer
* With/as
  + With open(“filename.txt”, “mode”) as variable:
  + Automatically closes file
  + Can execute operations inside of loop
* Open modes:
  + “w” write mode
  + “r” read only mode
  + “r+” read/write mode

**Importing**

* Import Library
  + Imports library
* From Library import Function
  + Imports specific function from a library
* as name
  + Add as at the end of import to import the thing under a custom name