## Operators

* Can be applied without math library - +, -, \*, /, %, \*\*
  + By default “/” always produces a floating point result, use “//” for integer result
  + Modulo operator % gives the remainder of a division
  + Example: 5/2 = 2.5; 5//2 = 2; 5%2 = 1
  + Two \*\* is exponential; e.g., 2\*\*16 = 65536; 9\*\*0.5 = 3
* Python honors PEMDAS order of operations
  + Parentheses, exponent, multiplication, division, addition, subtraction
* Bit wise operators
  + bin() for binary, e.g., bin(3) = 0b11
* & AND, | OR, ^ XOR, e.g. a & b, a | b, a ^ b
* Bit shifting <<, >>
* ~ ??????

## Logic

* Default flow is top to bottom; can repeat backwards, or skip forward in your code
  + if statement (skip forward) only run if condition is met
  + else statement is run if the “if condition” is not met
  + These statements can be nested
  + elif statements can be used to prevent excessive nesting

|  |
| --- |
| a = 6  if a==1:  print('one') elif a==2:  print('two') elif a==3:  print('three') elif a==4:  print('four') else:  print('No match')  print ('>>DONE') |

* Boolean operator - ==
  + Example:

|  |
| --- |
| * a = 5  if a==5: # equal to five  print('equal to five')  if a<=6: # less than 6  print('less than six')  if a>=4: # greater than 4  print('greater than four') |

* + Can be used with string operators as well
* Logical operators – AND, OR
  + AND is evaluated first
  + A true and false will produce a false
* An integer of 0 is false, any other number is true
* A blank string is false, any other string is true

## Loops

* Repeat a section of code
* Only loop construct is “while” – evaluated constantly if the condition is met
  + Use “a = a + 1” to increment an integer to escape the loop

|  |
| --- |
| a = 0  while a<10:  a = a + 1  if a == 10:  print('Made it to ten! ',a)  break # takes you out of the loop # continue # takes you back to the top of the loop  print('Not ten yet...',a) # not run if continue is executed from above line |

* Can use “while True:” can be used to have a loop run until it is manually exited with break or continue

|  |
| --- |
| * while True:   g = input('Still going? ')  if g=='no':  print('Ok bye!')  break  print('ok... carry on') |

* Iterator handler – “for” – NOT the same as a for loop in other languages – “for i in <object>” – it behaves like a “for each” loop
  + For example, if object is a string, it receives one letter at a time
  + Can also use the range function, e.g., “for i in range(<integer>)”
    - Can start in the middle of a range, e.g., “range(0-5,2)”
    - Can adjust the increment value, e.g. “range(0-5,2,0.5)”
* If using nested while statements, a “continue” or “break” will only go back to the most recent “while”
* Cannot break out of an “if” statement, “break” and “continue” only work with loops

## Lists and Tuples

* A list object lets you hold things, like an array or collection
* Can use brackets instead of built-in list functions, i.e., p[2] instead of p.\_\_getitem\_\_(2)
* Use brackets to reference a specific object from the list – i.e., p[3]
* A tuple is a list that you can’t change
* Created with parentheses instead of brackets
* Packing and unpacking can be performed by using multiple variables when referencing a list
  + Has to be exact, less or more variables will result in an error