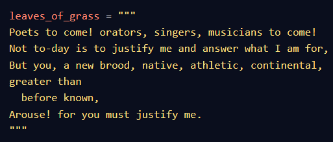
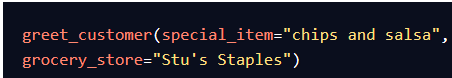
**Python 3.**

**Summary:**

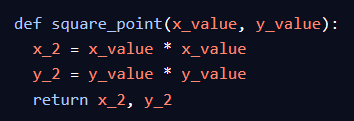
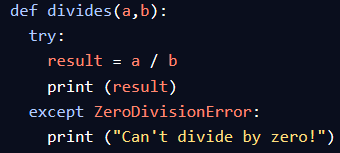
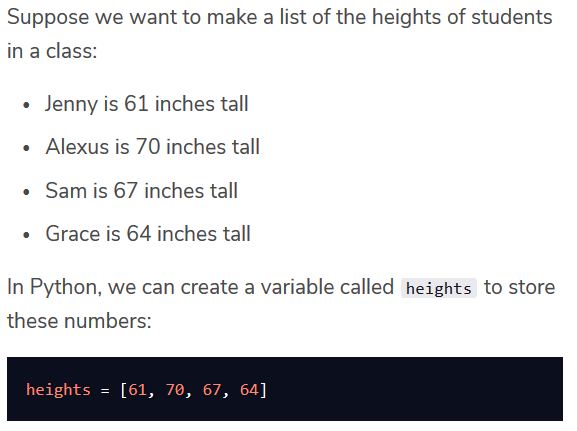
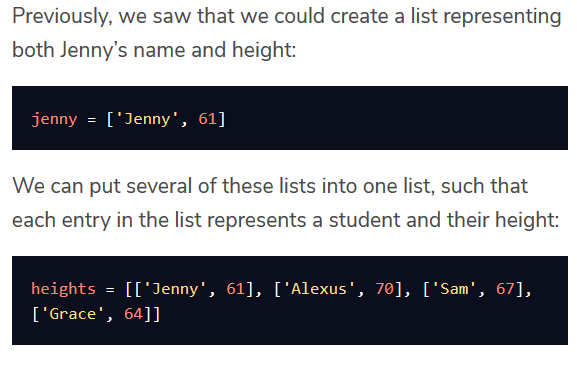
* General purpose versatile popular programming language. Concise and easy to read, also good language to have in programmer stack as can be used for everything from web dev to software dev and data science

**SYNTAX**

* Comments
  + Text written in prog, but not run by cpu called comment.
  + Python interprets anything after a # as a comment.
  + Provides context for why something is written way it is
* Spacing
  + In python amount of whitespace tells cpuu what is part of a function and what is not part of it.
  + Tabs are not the same as 2 spaces or 4 by default.
  + Stay consistent with using tabs or spaces
* Print
  + print( “whatever u want in here” )
  + print can print anything, no need to convert to string
  + unless doing mix of str and another datatype
* Strings
  + Blocks of text.
  + Can be surrounded with double or single quotes, **JUST BE CONSISTANT**
  + Can create string with repeated string by doing
    - String a=anotherString\*3; #repeats anotherstring 3 times in a
  + **Multi-Line Strings**
    - If want to write string spanning multiple lines use ‘’’ or “””
    - This lets compiler know string doesn’t end till next ‘’’.
* Variables
  + Way to store data for reuse. Assign variables with = sign in python.
    - Example
      * message\_string=”Hello There”
      * print(message\_string)
  + After var assigned to initial val, it CAN be reassigned to new val with dif data type from initial.
  + Can check type of a var using the type(var\_name) and print to get result displayed
* Errors
  + When error occurs, python will point it out with a ^ character.
  + When prog throws error we don’t expect to encounter, call those BUGS.
  + TWO COMMON ERRORS
    - SyntaxError:
      * Something wrong with way your prog written
        + Punctuation does not belong
        + Command where not expected
        + Missing parenthesis
    - NameError
      * Python Interpreter sees word does not recognize.
      * Code that contains something that looks like var but was never defined will throw error
* Numbers
  + Python has few numeric data types.
  + Int
    - Whole number, no decimal point and contains counting numbers as well as negative and the number 0
    - An\_int=2
  + Floating-Point number (float)
    - Decimal number. Can be used to rep fractional quantities.
      * Ex: average test score, length of wall…
      * A\_float=2.1
* Calculations
  + Performs standard +, - , \* , /
  + When doing devision, result has a decimal place. This is because Pyton converts all ints to float before performing division. (2.8 and up) version
  + DIVISION CAN THROW ZERODIVISIONERROR
  + EXPONENTS
    - Use the notation \*\*
    - Example: print(2 \*\* 10) # 2 to the 10th power
  + Modulo
    - %
    - Give remainder of division calc
    - If number divisible, then modulo operator will be 0
      * Ex: 29%5=4
      * 25%5=0
  + CONCATENATION (+)
    - Doesn’t just add numbers, CAN ADD STRINGS
    - Concatenating strings can create brand new string compromised of contents of first and second string…
    - IF WANT TO CONCATENATE A STRING WITH A NUM HAVE TO MAKE NUM A STRING FIRST USING str()
    - Ex:
      * Age=10;
      * Print(“I am” + str(age)+” years old”)
  + PLUS EQUALS
    - Shorthand for updating vars.
    - When have num saved in var and want to add curr value of var can use +=
    - Example:
      * Num\_hiked=12
      * numHiked+=2;
    - CAN BE UESD FOR STRING CONCATENATION TOO
* **Input**
  + Can assign val to var through user input.
  + We use keyword input()
    - Example:
      * Like\_snakes=input(“Do you like snakes”)
* **Functions** 
  + Usually created if some process or calculation have to be repeated.
  + Can save time from having to rewrite code
  + General Format:
    - def function\_name() :
      * #code here
  + PARAMETERS
    - Vars that can pass into function
    - Can do this
    - def function\_name(par1,par2…):
    - These called positional arguments…assignment depends on order placed in
    - Can use keyword arguments where explicitly refer to what each arg is assigned to in function call
    - Ex:
    - Can also define default args for function so if nothing placed inside, has defaukt.
      * Example:



NOTE: KEYWORD ARGUMENTS WRITTEN SO THEY COME AFTER ALL POSITIONAL ARGUMENTS IN FUNCTION DEFINITION

* + RETURNS
    - Functions can return a val to user so that this val can be modified or used later using “return” keyword
    - Can return multiple return values by separating them with a comma
      * 
    - Can get these values by doing:
      * X\_squared,y\_squared=square\_point(1,3)
      * When returning mult vals separated by commas, all vals listed actualy wrapped in a tuple (1,2,3…) that tuple is returned containing each val.
      * Can tuple unpack where if tuple has mult vals, can assign same num of vars to number of elements in tuple.
* **Boolean Expressions**
  + To add control flow to program, want to eval if statement true or not.
  + Can assign true or false to vars using var\_name=True… # or False
  + Notice first letter capitalized
  + Relational operators:
    - == #equals, compares two items return true or flase
    - != #not equals
    - > …….. >=
    - <………<=
  + Boolean Operators
    - and
    - or
    - not #this is actual word, not ! like in Java
  + Conditional Statements
    - If(some condition):
      * Do stuff…
    - Elif(some condition):
      * Do stuff
    - Else:
      * Do stuff
    - TRY EXCEPT STATEMENS
      * try:
        + #some statement
      * except ErrorName:
        + #some statement
      * Statement under try will execute, if there is an exception then goes to except statement and executes that and try terminates.
    - Can raise errors using “raise” keyword
* **Lists**
  + Ordered set of objects in Python
  + Can make list of anything, even combine multiple data types
    - Even list of lists
  + ZIP
    - If want to create list of lists that paired together  
      , can use zip.
    - EXAMPLE:
      * Name\_and\_heights=zip(names,heights)
      * #names and heights are each a ist.
      * #can print out by casting to list print(list(name\_and\_heights))
    - Can create empty list be leaving empty brackets
    - Can add element to list using .append()
    - Ex:
      * Empty\_list.append(1) #adds to end of list
    - CAN combine two lists using +
    - Ex: added\_lists=list1+list2
    - Range
      * If want to create list of consecutive nums.
        + Ex: my\_range=range(10)
        + Gens number from 0-> n-1
      * If pass in two args, can create list starting at dif nums.
        + Ex: range(2,9)
      * If pass in 3 args, can create diff starting, and interval number
        + Ex: range(2,9,2) will skip increments of 2 from 2-> 8
    - Length
      * Can find length of list using len keyword
        + Ex: print(len(my\_list))
    - Indexes
      * Python is zero indexed
      * Can access each element in list using list[index#]
      * **Can use index -1 to select last item of list**
    - Slice
      * can slice list if don’t want all of it using list
        + ex: sublist=letters[1:6]
        + print(sublist) //will print to index-1 so if want index 6 need to do letters[1:7]
      * If want to select last few elements of a list can do:
        + Print(fruits[-3:]) for example
    - Count
      * If want to know how many times something appears in list can use count function
        + Ex: letters.count(‘I’)
    - Sorting
      * If want to sort numerical or alphabetical can use .sort()
      * Sort does not return anything
      * Can also use sorted(list) this returns a sorted list instead so the original list stays the same
  + Tuples
    - DS in python, allowing store mult data inside it.
    - VERY SIMILAR TO LIST, but IMMUTABLE
      * Once create, elements,order,how many they are cant be changed.
    - Ex: my\_inf=(‘Mike’,24,’Programmer’) 🡸 tuple
    - Can be used like lists, if want to access info can do
      * My\_info[0]… etc.
      * Tuples immutable, so can’t change! Can’t add or remove.
      * What u see is what u get.
    - Can unpack….
      * What this means is that the info in tuple can be stored in var.
      * Ex: name,age,occupation=my\_info
      * So it gets the info from my\_info tuple and store into these created variables.
      * As long as number of vars created for elements in tuple, can do this!
    - SPECIAL CASE
      * Creating one element tuple
      * Ex: one\_elmeent\_tuple=(4)
      * Notice we only get value 4, it doesn’t create tuple
      * THIS BECAUSE in math can surround num in parenthesis like math….
      * SO IF WANT TO CRAETE ONE ELEMENT TUPLE,
      * Have to do
        + One\_elmeent\_tuple(4,)
        + NOW itll be a tuple.
    - When to use?
      * Store data meant to be together but not necessarily similar.
      * So example the data like my\_info describes me so it should be together…..
      * Order MATTERS FOR tuples, so unpacking for example has to be in order…
  + Loops
    - Iteration especially through lists
    - General format
      * For <temp\_var name> in <list var name>:
        + <do some actions>
  + **Can use in range to iterate through certain amount**
    - Ex: for I in range(num):
      * Do stuff
  + Infinite loops defined as:
    - For temp\_var\_name in list\_name:
      * List\_name.append(something)
        + List will never end since appending so never terminate
  + Can use break keyword to break out of loop early
  + Continue keyword can be used to skip values and move to next iteration
* While Loops
  + While(sum condition):
    - Do stuff
  + If have list of lists, can iterate by doing:
    - for temp\_name\_1 in list\_name:
      * for(temp\_name\_2 in temp\_name\_1:
        + do stuff
* **List Comprehension**
  + 

