**Python Notes**

*Constants*

* Numeric constants are numbers
* print(123)
* String constants use single (‘) or double (“) quotes
* print(‘Hello World’)

*Reserved Words*

* False
* None
* True
* and
* as
* assert
* break
* class
* if
* def
* del
* elif
* else
* except
* return
* for
* from
* global
* try
* import
* in
* is
* lambda
* while
* not
* or
* pass
* raise
* finally
* continue
* nonlocal
* with
* yield

*Variables*

* A named place in the memory where a programmer can store data and later retrieve the data using the variable ‘name’.
* Programmers choose the names of the variables.
* You can change the contents of a variable in a later statement.
* Variable names must start with a letter or underscore \_
* Variable names must consist of letters, numbers and underscores.
* Variable names are case sensitive.
* x = 12.2
* y = 14

*Assignment Statements*

* We assign a value to a variable using the assignment statement (=)
* x = 1

*Numeric Expressions (Operators)*

* + addition
* - subtraction
* \* multiplication
* / division
* \*\* power
* % remainder

*Operator Precedence Rules*

1. Parenthesis
2. Exponentiation
3. Multiplication, Division and Remainder
4. Addition and Subtraction.
5. Left to right.

*String Conversions*

* Python will print values that have been multiplied or divided with a decimal point.
* Type() tells you the type of value (string, integer etc.)
* Float()
* Int()
* You can use int() and float() to convert between strings and integers (numbers)

*User Input*

* We can instruct Python to pause and read data from the user using the input() function.
* The input() function returns a string.

*Comments in Python*

* Anything after a # is ignored by Python.

*Conditional Execution*

*Conditional Steps*

x = 5

if x < 10:

print(‘Smaller’)

if x > 10:

print(‘Bigger’)

print(‘Finis’)

*Comparison Operators*

* Boolean expressions ask a question and produce a Yes or No result which we use to control program flow.
* Comparison operators look at variables but do not change the variables.
* < less than
* <= less than or equal to
* == equal
* >= greater than or equal to
* > greater than
* != not equal

*Indentation*

* Increase indent after an if statement or for statement (after : )
* Maintain indent to indicate the scope of the block (which lines are affected by the if/for).
* Reduce indent back to the level of the if statement or for statement to indicate the end of the block.
* Blank lines are ignored – they do not affect indentation.
* Comments on a line by themselves are ignored with regard to indentation.
* Don’t use tabs, use four spaces for indentation.
* Increase/maintain after if or for.
* Decrease to indicate end of block.

*The try / except Structure*

* You surround dangerous section of code with try and except.
* If the code in the try works – the except is skipped.
* If the code in the try fails – it jumps to the except section.

*Functions*

* def = define function.
* def stores/remembers the code.
* def thing() :

print (‘Hello’)

print (‘Fun’)

* Two kinds of functions in Python.
* Built in functions – provided as part of Python – print(), input(), type(), float()
* Functions that we define ourselves and then use.
* We treat the built in function names as “new” reserved words (we avoid them as variable names).

*Function Definition*

* A function is some reusable code that takes arguments as input, does some computation and then returns a result or results.
* We define a function using the def reserved word.
* We call/invoke the function by using the function name, parenthesis and arguments in an expression.
* A function is some stored code that we use.
* A function takes an input and produced an output.

*Building our Own Functions*

* We create a new function using the def keyword followed by optional parameters in parentheses.
* We indent the body of the function.
* This defines the function but does not execute the function.

*Definitions and Uses*

* Once we have defined a function we can call or invoke it as many times as we like.
* This is the store and reuse pattern.

*Arguments*

* An argument is a value we pass into the function as its input when we call the function.
* We use arguments so we can direct the function to do different kinds of work when we call it at different times.
* We put the arguments in parentheses after the name of the function.
* big = max(‘Hello world’)

*Parameters*

* A parameter is a variable which we use in the function definition.
* It is a ‘handle’ that allows the code in the function to access the arguments for a particular function invocation.

*Return Values*

* Often a function will take its arguments, do some computation, and return a value to be used as the value of the function call in the calling expression.
* The return keyword is used for this.
* A fruitful function is one that produces a result (or return value).
* The return statement ends the function execution and ‘sends back’ the result of the function.

*Multiple Parameters / Arguments*

* We can define more than one parameter in the function definition
* E.g. def addtwo (a, b) :
* We simply add more arguments when we call the function.
* We match the number and order of arguments and parameters.
* Parameters are the names listed in the function’s definition.
* Arguments are the real values passed to the function.
* When a function does not return a value we call it a ‘void’ function.
* Functions that return values are fruitful functions.

*Loops and Iterations*

* Loops (repeated steps) have iteration variables that change each time through a loop.
* Often these iteration variables go through a sequence of numbers.

*Breaking Out of a Loop*

* The break statement ends the current loop and jumps to the statement immediately following the loop.
* It is like a loop test that can happen anywhere in the body of the loop.

*Finishing an Iteration with continue*

* The continue statement ends the current iteration and jumps to the top of the loop and starts the next iteration.

*Indefinite Loops*

* While loops are called ‘indefinite loops’ because they keep going until a logical condition becomes False.
* The loops we have seen so far are pretty easy to examine to see if they will terminate or if they will be ‘infinite loops’.
* Sometimes it is a little harder to be sure if a loop will terminate.

*Definite Loops*

* Quite often we have a list of items of the lines in a file – effectively a finite set of things.
* We can write a loop to run the loop once for each of the items in a set using the Python ‘for’ construct.
* These loops are called ‘definite loops’ because they execute an exact number of times.
* We say that definite loops iterate through the members of a set.
* Definite loops (for loops) have explicit iteration variables that change each time through a loop.
* These iteration variables move through the sequence or set.

*Looking at In*

* The iteration variable iterates through the sequence (ordered set).
* The block (body) of code is executed once for each value in the sequence.
* The iteration variable moves through all of the values in the sequence.

*Counting in a Loop*

* To count how many times we execute a loop, we introduce a counter variable that starts at 0 and we add one to it each time through the loop.

*Summing in a Loop*

* To add up a value we encounter in a loop, we introduce a sum variable that starts at 0 and we add the value to the sum each time through the loop.

*Finding the Average in a Loop*

* An average just combines the counting and sum patterns and divides when the loop is done.

*Filtering in a Loop*

* We use an if statement in the loop to catch / filter the values we are looking for.

*Search Using a Boolean Variable*

* If we just want to search and know if a value was found, we use a variable that starts at False and is set to True as soon as we find what we are looking for.

*Finding the Smallest Value*

* The first time through the loop, the smallest is None, so we take the first value to be the smallest.

*The ‘is’ and ‘is not’ Operators*

* Python has an is operator that can be used in logical expressions.
* Implies ‘is the same as’.
* Similar to but stronger than ==
* Is not also is a logical operator.
* Use is sparingly.

*Strings*

* A string is a sequence of characters.
* A string uses quotes.
* For strings + means concatenate.
* When a string contains numbers, it is still a string.
* We can convert numbers in a string into a number using int()

*Reading and Converting*

* We prefer to read data in using strings and then parse and convert the data as we need.
* This gives us more control over error situations and or bad user input.
* Raw input numbers must be converted from strings.

*Looking Inside Strings*

* We can get at any single character in a string using an index specified in square brackets.
* The index value must be an integer and starts at zero.
* The index value can be an expression that is computed.

*Strings Have Length*

* The built in function len() gives us the length of a string.

*Looping Through Strings*

* Using a while statement and an iteration variable and the len() function we can construct a loop to look at each of the letters in a string individually.
* A definite loop using a for statement is much more elegant.
* The iteration variable is completely taken care of by the for loop.

*Looking Deeper into in*

* The iteration variable iterates through the sequence.
* The block of code is executed once for each value in the sequence.
* The iteration variable moves through all of the values in the sequence.
* The iteration variable iterates through the string and the body of code is executed once for each value in the sequence.

*Intermediate Strings*

*Slicing Strings*

* We can look at any continues section of a string using a colon operator.
* The second number is one beyond the end of the slice – up to but not including.
* If the second number is beyond the end of the string it stops at the end.
* If we leave off the first number or the last number of the slice, it is assumed to be the beginning or end of the string respectively.
* s = ‘Monty Python’

print(s[0:4])

Mont

*String Concatenation*

* When the + operator is applied to strings it means concatenation.
* a = ‘Hello’

b = ‘There

c = a + ‘ ‘ + b

print(c)

Hello There

*Using in as a logical Operator*

* The in keyword can also be used to check to see if one string is ‘in’ another string.
* The in expression is a logical expression that returns True or False and can be used in an if statement.
* fruit = ‘banana’

‘nan’ in fruit

True

* if ‘a’ in fruit :

print(‘Found it!’)

Found it

*String Library*

* Python has a number of string functions which are in the string library.
* These functions are already built into every string – we invoke them by appending the function to the string variable.
* These functions do not modify the original string, instead they return a new string that has been altered.
* greet = ‘Hello Bob’

zap = greet.lower()

print(zap)

hello bob

*Searching a String*

* We use the find() function to search for a substring within another string.
* find() finds the first occurrence of the substring.
* If the substring is not found, find() returns -1.
* Remember that string position starts at zero.
* fruit = ‘banana’

pos = fruit.find(‘na’)

print(pos)

2

*Changing Upper and Lower Case*

* You can make a copy of a string in lower case or upper case.
* Often when we are searching for a string using find() we first convert the string to lower case so we can search a string regardless of case.
* greet = ‘Hello Bob’

nnn = greet.upper()

print(nnn)

HELLO BOB

* www = greet.lower()

print(www)

hello bob

*Search and Replace*

* The replace() function is like a search and replace operation.
* It replaces all occurrence of the search string with the replacement string.
* greet = ‘Hello Bob’

nstr = greet.replace(‘Bob’, ‘Jane’)

print(nstr)

Hello Jane

*Stripping Whitespace*

* Sometimes we want to take a string and remove whitespace at the beginning and or end.
* lstrip() and rstrip() remove whitespace at the left or right.
* strip() removes both beginning and ending whitespace.

*Prefixes*

* line = ‘Please have a nice day’

line.startswith(‘Please’)

True

* line.startswith(‘p’)

False

*Reading Files*

*File Processing*

* A text file can be thought of as a sequence of lines.

*Opening a File*

* Before we can read the contents of the file, we must tell Python which file we are going to work with and what we will be doing with the file.
* This is done with the open() function.
* open() returns a ‘file handle’ – a variable used to perform operations on the file.

*Using open()*

* handle = open(filename, mode)
* Returns a handle use to manipulate the file.
* Filename is a string.
* Mode is optional and should be ‘r’ if we are planning to read the file and ‘w’ if we are going to write to the file.
* fhand = open(‘mbox.txt’, ‘r’)

*The newline Character*

* We use a special character called the ‘newline’ to indicate when a line ends.
* We represent it as \n in strings.
* Newline is still one character not two.
* A text file has newlines at the end of each line.
* stuff = ‘Hello\nWorld!’

print(stuff)

Hello

World!

*Files as a Sequence*

*File Handle as a Sequence*

* A file handle open for read can be treated as a sequence of strings where each line in the file is a string in the sequence.
* We can use the for statement to iterate through a sequence.
* xfile = open(‘mbox.txt’)

for cheese in xfile :

print(cheese)

*Counting Lines in a File*

* Open a file read only.
* Use a for loop to read each line.
* Count the lines and print out the number of lines.
* fhand = open(‘mbox.txt’)

count = 0

for line in fhand :

count = count + 1

print(‘Line Count: ’, count)

*Reading the Whole File*

* We can read the whole file (newlines and all) into a single string.
* fhand = open(‘mbox.txt’)

inp = fhand.read()

print(len(inp))

94626

*Searching Through a File*

* We can put an if statement in our for loop to only print lines that meet some criteria.
* fhand = open(‘mbox.txt’)

for line in fhand :

if line startswith(‘From:’) :

print(line)

*Blank Lines*

* Each line from the file has a newline at the end.
* The print statement adds a newline to each line.

*Searching Through a File (fixed)*

* We can strip the whitespace from the right hand side of the string using rstrip() from the string library.
* The newline is considered white space and is stripped.
* fhand = open(‘mbox-short.txt’)

for line in fhand :

line = line.rstrip()

if line.startswith(‘From’) “

print(line)

*Skipping with continue*

* We can conveniently skip a line by using the continue statement.
* fhand = open(‘mbox-short.txt’)

for line in fhand :

line = line.rstrip()

if not line.startswith(‘From:’) :

continue

print(line)

*Using in to select lines*

* We can look for a string anywhere in a line as our selection criteria.

*Programming*

*Algorithms*

* A set of rules or steps used to solve a problem.

*Data Structures*

* A particular way of organising data in a computer.

*Not a Collection*

* Most of our variables have one value in them.
* When we put a new value in the variable the old value is overwritten.

*Collection (Lists)*

* A collection allows us to put many values in a single variable.
* A collection is nice because we can carry all many values around in one convenient package.
* friends = [‘Joseph’, ‘Glenn’, ‘Sally’]

*List Constants*

* List constants are surrounded by square brackets and the elements in the list are separated by commas.
* A list element can be any Python object, even another list.
* A list can be empty.
* For loops.

*Looking Inside Lists*

* Just like strings, we can get at any single element in a list using an index specified in square brackets.
* Starts at 0.

*Lists are Mutable*

* Strings are immutable. We cannot change the contents of a string, we must make a new string to make any change.
* Lists are mutable. We can change an element of a list using the index operator.

*How Long is a List*

* The len() function takes a list as a parameter and returns the number of elements in the list.
* Actually len() tells us the number of elements in any set or sequence (such as a string).

*Using the Range function*

* The range function returns a list of numbers that range from zero to one less than the parameter.
* We can construct and index loop using for an integer iterator.

*Concatenating lists using +*

* We can create a new list by adding two existing lists together.

*Lists can be sliced using :*

* Just like in strings, the second number is 'up to but not including’.
* t = [9, 41, 12, 3 74, 15]

t[1:3]

[41, 12]

*Building a List from Scratch*

* We can create an empty list and then add elements using the append() method.
* The list stays in order and new elements are added at the end of the list.
* stuff = list()

stuff append(‘book’)

stuff append(99)

print(stuff)

[‘book’, 99]

*Is Something in a List*

* Python provides two operators that let you check if an item is in a list.
* In and not in.
* These are logical operators that return True or False.
* They do not modify the list.

*Lists are in Order*

* A list can hold many items and keeps those items in the order until we do something to change the order.
* A list can be sorted.
* The sort() method means ‘sort yourself’.

*Built in Functions and Lists*

* There are a number of functions built into Python that take lists as parameters.
* Len()
* Max()
* Min()
* Sum()

*Strings and Lists*

* Split() breaks a string into parts and produces a list of strings.
* We think of these as words.
* We can access a particular word or loop through all the words.
* You can specify what delimiter character to use in the spacing.
* line = ‘first;second;third’

thing = line.split(‘;’)

print(thing)

[‘first’, ‘second’, ‘third’]

print(len(thing))

3

*The Double Split Pattern*

* Sometimes we split a line on way and then grab one of the pieces of the line and split that piece again.

*Python Dictionaries*

*What is a Collection?*

* A collection is nice because we can put more than one value in it and carry them all around in one convenient package.
* We have a bunch of values in a single variable.
* We do this by having more than once place ‘in’ the variable.
* We have ways of finding the different places in the variable.

*What is not a Collection?*

* Most of our variables have one value in them.
* When we put a new value in the variable the old value is overwritten.

*A List*

* A linear collection of values that stay in order.
* Lists index their entries based on the position in the list.

*A Dictionary*

* A bag of values each with its own label.
* Dictionaries are like bags – no order.
* So we index the things we put in the dictionary with a ‘lookup tag’.
* Dictionaries are python’s most powerful data collection.
* They allow us to do fast database like operations in python.
* Dictionaries are like lists except that they use keys instead of numbers to look up values.
* They have different names in different languages.
* Associative arrays – perl / php
* Properties or Map or HashMap – Java
* Property Bag – C# / .Net
* purse = dict()

purse[‘money’] = 12

purse[‘candy’] = 3

purse[‘tissue’] = 75

print(purse)

[‘money’ : 12, ‘tissues’: 75, ‘candy’ : 3)

*Dictionary Literals (Constants)*

* Dictionary literals use curly brackets and have a list of key : value pairs.
* You can make an empty dictionary using empty curly brackets.
* jjj = {‘chuck’ : 1 , ‘fred’ : 42, ‘jan’ : 100}

print(jjj)

{‘jan’ : 100, ‘chuck’ : 1, ‘fred’ : 42}

* ooo = { }

print{ooo}

{ }