POP

Principles of Programming

Assignment statement

Creating a new variable and giving it a value. Variable names can only be created with A-Z, a-z, 0-9 and underscore _. Variable names cannot start with 0-9 or _ must only start with A-Z and a-z.

new_variable = 76

It conventional to use lower case for variable names. You will get an syntax error if you start a variable name with a number

To input data into python, use **input ()**

Numbers are represented as **int** to convert to an integer use **int ()**

Text is presented as **str** to convert to a string use **str ()** or use single ‘ ‘ or double quotes “ ”

‘5‘ + ‘5’ = 55 not 5 as + is treated as **concatenation** with strings

‘Spam‘ *3 can used to perform string repetition, means ‘Spam‘ + ‘Spam‘ + ‘Spam‘ which gives SpamSpamSpam

Strings are **immutable** objects meaning they can be altered or changed!

To print numbers and strings together must convert numbers to strings!

Len() functions returns how may characters in a string.

For and while loop on string

```
>>> furit = 'bannana'
>>> index = 0
>>> while index < len(furit):
        letter = furit[index]
        print(letter)
        index += 1


b
a
n
n
a
n
a
```

```
>>> for letter in furit:
        print(letter)


b
a
n
n
a
n
a
```

Slices - string single character given from S[i] where i is a character of string also called an index, start counting from 0. If index is negative it counts backwards.

String S	h	e	l	l	o

index	S[0]	S[1]	S[2]	S[3]	S[4]
Index	S[-5]	S[-4]	S[-3]	S[-2]	S[-1]

Substring – slice has two apartments S[a:b] give a substring of length b – a, starting with character a and ending until character b but not including it. Negative number can be used. If you omit the second parameter and leave colon, the slice goes to the end of the string. Likewise if you omit the first parameter the slice goes from the beginning of string. No parameters just colon, S[:] just gives S. String are immutable, they cannot be changed.

Subsequence – slice with three parameters S[a:b:c], the third is a step. If positive it increments by c. If negative -1 goes in reverse order.

String methods – **find()** is a search method, s.find('E') searches for E in the string S. If found returns the index of the first occurrence if not found method returns -1.

While **rfind()** returns the index of last occurrence.

If you call find() with three arguments **s.find(substring, left, right)**, the search is performed inside the slice **s[left:right]**. If you specify only two arguments, like s.find(substring, left), the search is performed in the slice **s[left:],** that is, starting with the character at index left to the end of the string.

String methods – **replace()** is a replace method, **s.replace(old, new)** takes the string S and replaces all occurrences of substring old with the substring new. One can pass the third argument count, like this: s.replace(old, new, count). It makes replace() to replace only first count occurrences and then stop.

These methods don't change s, they return a new string:

s.isupper() and s.islower() tests if letters in s are uppercase or lowercase, respectively
s.upper() and s .lower() converts all letters in s to uppercase or lowercase, respectively
s.isalpha() tests if all characters in s are alphabetic
s.isdigit() tests if all characters in s are digits
s.isspace() tests if all characters in s are *whitespace*
whitespace includes spaces, tabs, newlines, and a few other nonprinting characters
s.lstrip(), s.rstrip(), s.strip() removes whitespace from the left end, the right end, or both ends
s .startswith(substring), s .endswith(substring) test whether s starts with, or ends with, substring.
sep .join(sequence) inserts the string sep between the elements of sequence , and returns a new string

format method

print("%c is my %s letter and my number %d number is %.5f" % ('X', 'favorite', 1, .14))

The **format** method looks for braces, **{}**, in a string, and substitutes values for those braces

· **format** is mostly used to prepare a string for printing, and often occurs within a call to

print

· We will cover only the simplest cases of this very complex method

· Simple substitution, in order:

String Methods

find	**rfind**	**upper**	**isupper**	**format**	**isdigit**
replace	**sort**	**lower**	**islower**	**isspace**	**isalpha**

```
>>> print('a bar is a bar, essentially'.replace('bar', 'pub', 1))
a pub is a bar, essentially
```

String methods – **count()** is a count method, **s.count(substring)** counts the number of occurrences of one string substring within another string s

```
>>> print('Abracadabra'.count('a'))
4
```

Numbers can also be presented as real numbers or floating-point number **float** to convert to floating point use **float ()**

Other casts include:

hex() gives string representing the hexadecimal value --base 16, 0 to 9, A to F

oct() gives string representing the octal value --base 8, 0 to 7

bin() gives string representing the binary value --base 2, 0 to 1

chr() gives character represented the Unicode value

ord() gives interger value represented the Unicode character

Bitwise operators:

- **~** is bitwise not
- **&** is bitwise and
- **|** is bitwise or
- **^** is bitwise exclusive xor
- **x >> i** shifts the bits in x to the right i places
- **x << i** shifts the bits in x to the left i places

Python has the following seven mathematical operators:

- Addition 5 + 7 gives 12
- Subtraction 5 – 3 gives 2
- Multiplication 5 * 3 gives 15
- Division 4 /3 gives 1.3333333333333333
- Floor Division 5 // 3 (always rounds down) gives 1
- Exponent 3 ** 2 (3^2) gives 9
- Modulus 5 % 2 (gives remainder) gives 1

This operation also has a order of operation given my **PEMDAS**

- Parentheses have the highest precedence, forces expression to do what's in the brackets first!
- Exponentiation has the next highest precedence, 1 + 3**3 = 28 not 64
- Multiplication and division have the higher precedence than addition and subtraction, 6 + 6*2 = 18 not 24
- Operators with the same precedence are evaluated from left to right (except exponentiation). Use parentheses to make it obvious if unsure.
- Parenthesis (), Brackets [], braces {}

The function **round ()** is also available**,** if one argument is present rounds to nearest integer, if two arguments present rounds to the specifies number of digits

Round(2.4566) gives 2

Round(2.4566, 2) gives 2.46

Another inbuilt function is absolute value, **abs ()**

Ex abs(-3) gives 3

Ex abs(3) gives 3

math module

Import module by writing the following instruction at start of script

import math

It allows you to import other functions to use in your programme

floor(x) gives the floor of x, the largest integer less than x

ceil(x) gives the ceiling of x, the smallest integer great than or equal to x

sqrt(x) gives the square root of x

log(x) with one argument, it gives log of x to the base e, with two argument it gives log of x to the given base.

e gives mathematical constant 2 = 2.71828

sin(x) gives the sine of x in radians

asin(x) fives arcsine of x in radian

pi gives the mathematical constant

j is the imaginary number i, doesn't use i cause engineers use it for something else

import math

Is the statistics model for mean, median and mode, stdev and variance

Conditionals

If something is true:

Do something

Elif something else is true:

Do this

Else:

Do this if both above is false

```
def ex4(a, b, c):
    if a == b == c:
        print(3)
    elif a != b and b != c and a !=c:
        print(0)
    else:
        print(2)
```

Both if, elif and else end with a colon, **:**

And the next line after colon is indented with 4 spaces of one tab

The following operators can be used in condition statements

< less than

> greater than

<= less or equal, remember no such thing as =>

>= greater or equal, remember no such thing as =<

== equal

! = not equal

Chained operations may be used

A == B == C

X <= Y <= Z

Hash

The hash symbol # is used to comment out text within the code so it doesn't affect the programme.

Boolean expressions

True and **False** are special types called bool, use type () to see

Debugging

Python gives three kinds of errors, syntax errors, runtime errors and semantic errors.

→ An error is a mistake in your code and it's your fault.
→ An exception is a mistake that isn't in your code and isn't your fault but can't be ignored

Syntax error refers to the structure of the programme and the rules about that structure

Runtime Error does not appear until after the programme has run, these errors are called exceptions. NameErrors, misspelt or not consistent, local variables. TypeErrors using non-integer index on list, string or tuple. IndexError when you go outside the index range of a string, list, tuple. KeyError accessing an element of a dictionary that key doesn't contain. AttributeError check spelling n vars for what exists, if nonetype is none problem not attribute but object. Index error is array right size and index right value

Semantic Error will allow your programme to run without generating any error messages, but it will not do the right thing

Use **PRINT** statement through code to see if certain parts are working. **Assert** checked by computer by Boolean expression and gives Assertion Errors if False. Catch exception using try: , except something :, except something else:, finally:

```
def get_int(prompt):
    """Gets an integer from the user"""
    try:
        age = int(input(prompt))
        return age
    except ValueError:
        print("That's not an integer!")
        return get_int(prompt)
```

```
>>> get_int("What is your age? ")
What is your age? fh
That's not an integer!
What is your age? dffdf
That's not an integer!
What is your age? 99
99
```

Shorthand code example are

A = A + B	same as A += B
A = A - B	same as A -= B
A = A * B	same as A *= B

Loops can use loops to repeat things using the for and while loops

```
for i in range(a,b):
                print(u)

names = [ “Tom”, “Barry”, “Ben”]
for name in names:
        print(name)
```

Range Function range (min_value, max_value) generates a sequence with numbers min_value, min_value + 1, ..., max_value - 1. NB the last number is not included so pay attention that maximum value in range () is n + 1 to make i equal to n on the last step. When only one argument in range it is considered the max value and min value set to zero eg range(max_value). Use indentation to control what instructions are affected by for and which aren’t. To iterate over a decreasing sequence, we can use an extended form of range() with three arguments - range(min_value, max_value, step). When omitted, the step is implicitly equal to 1, third argument cannot be zero.

```
for i in range (10, 0, -2):
    print(i**2)

names = [ "Tom", "Barry", "Ben"]
for name in names:
    print(name)
```

while loop is simple and based on Boolean true false logic. If true statements executed if false does nothing

```
def countdown(n):
    while n > 0:
        print(n)
        n = n - 1
    print("Blastoff!")
```

break statement is used to exit loop early

```
for I in range (1, 6):
    if I == 4:
            break
```

continue statement is used to skip rest of loop and start from the top

```
for I in range (1, 6):
        if I == 4:
                continue
```

yield statement not covered

break, continue only makes sense within an if statement

pass statement is used as a placeholder where a statement is needed but not known yet

```
if illegal alien(candidate):
    pass
else:
    hire(candidate)
```

Print function by default prints all its arguments separating them by a space and the puts a newline symbol after it. The separator can be change with sep = and the end line can be change by end =

Functions to create a function of your own use def this creates a function or a series of
statements to be executed but doesn't run it, one function might be the starting point this, by convention this function is called main and usually has no parameters and is called by running main(). Use triple quotes to define what the function does, the function as many arguments as parameters function_name (argument1 , ..., argumentN). Scope of variable names in a function are by default local variables. Functions end in three ways with return being a value, None, end. A function is pure if the only info it gets is from its parameters and the only info it produces is the value in return.

A global variable is one whose scope is the entire program can complicate debugging and understanding of program. Make a variable global by declaring it outside of function or in function using global.

```
foo = 77

def bar():
    foo = 33
    print (foo)

bar()
#prints 33
```

```
foo = 77

def main():
    global foo
    print (foo)

main()
# prints 77
```

Nonlocal variables are used in nested functions to make it local in other functions but not the nested function

```
def average(x, y):
    sum = 0
    def add(x, y):
        nonlocal sum
        sum = x + y
    add(x, y)
    return sum / 2
# average(3,5) returns 4
```

```
def average2(x, y):
    sum = 0
    def add(x, y):
        sum = x + y
    add(x, y)
    return sum / 2
# average(3,5) returns 0
```

Code that appears after a return statement, or any other place the flow of execution can never reach, is called **dead code**.

Lambdas are anonymous functions because they are used as a parameter to some other function, **lambda arg_1, ..., arg_N: expression**

```
>>> lst = ['Betsy', 'goes', 'to', 'the', 'University',
'of', 'Pennsylvania']

>>> sorted(lst, key = lambda x: x.lower())

['Betsy', 'goes', 'of', 'Pennsylvania', 'the', 'to', 'University']
```

Lambda functions are used along with built-in functions like filter() and map().

Map() applies a function to all the items in an input list

```
def multiply(x):
    return (x * x)

def add(x):
    return (x + x)

func = [add, multiply]
for i in range(5):
    value = list(map(lambda x: x(i), func))
    print(value)
```

```
[0, 0]
[2, 1]
[4, 4]
[6, 9]
[8, 16]
```

Filter as the name suggests creates a list of elements for which the condition returns True

Version Control git Commands that are used in version control and git bash/git hub

cd ~ change directory uses double quotes not like CMD

```
Michelle Conway@LAPTOP-SUCEBC4G MINGW64 /i/POP (master)
$ cd "I:\POP\Worksheets\Volume-of-a-Cone-Calculator-mconwa02"
```

git clone ~ Makes a copy of a depository and save in a local directory location sometimes asks for username and password

```
Michelle Conway@LAPTOP-SUCEBC4G MINGW64 /i/POP (master)
$ git clone https://github.com/BBK-DCSIS-PoP-I-2017-18/Volume-of-a-Cone-Calculat
or-mconwa02.git
Cloning into 'Volume-of-a-Cone-Calculator-mconwa02'...
fatal: HttpRequestException encountered.
   An error occurred while sending the request.
Username for 'https://github.com': mconwa02
remote: Counting objects: 19, done.
remote: Total 19 (delta 0), reused 0 (delta 0), pack-reused 19
Unpacking objects: 100% (19/19), done.
Checking connectivity: 19, done.
```

git init ~ initialise git for the project

```
Michelle Conway@LAPTOP-SUCEBC4G MINGW64 /i/POP/Worksheets/Volume-of-a-Cone-Calcu
lator-mconwa02 (master)
$ git init
Reinitialized existing Git repository in I:/POP/Worksheets/Volume-of-a-Cone-Calc
ulator-mconwa02/.git/
```

git remote add origin http:/blahblah ~ no quotes git

```
Michelle Conway@LAPTOP-SUCEBC4G MINGW64 /i/POP (master)
$ git remote add origin https://github.com/BBK-DCSIS-PoP-I-2017-18/Volume-of-a-Cone-Calculator
-mconwa02.git
```

git pull origin master ~ pulls down the dictory from github to start working on

```
Michelle Conway@LAPTOP-SUCEBC4G MINGW64 /i/POP (master)
$ git pull origin master
```

git add . ~ adds all file changes use commit straight after

```
Michelle Conway@LAPTOP-SUCEBC4G MINGW64 /i/POP/Worksheets/Volume-of-a-Cone-Calculat
or-mconwa02 (master)
$ git add .
```

git add filename ~ adds changes of named file use commit straight after

```
Michelle Conway@LAPTOP-SUCEBC4G MINGW64 /i/POP/Worksheets/Volume-of-a-Cone-Calculat
or-mconwa02 (master)
$ git add TestConeVolumeCalculator.py
```

git commit -m “I added such and such”

```
Michelle Conway@LAPTOP-SUCEBC4G MINGW64 /i/POP/Worksheets/Volume-of-a-Cone-Calculat
or-mconwa02 (master)
$ git commit -m "update test code"
[master 7ed7852] update test code
 1 file changed, 20 insertions(+)
```

git status ~ shows files that have had changes, red deleted/added, or tree clean

```
Michelle Conway@LAPTOP-SUCEBC4G MINGW64 /i/POP/Worksheets/Volume-of-a-Cone-Calculat
or-mconwa02 (master)
$ git status
On branch master
Your branch is up-to-date with 'origin/master'.

nothing to commit, working tree clean
```

```
Michelle Conway@LAPTOP-SUCEBC4G MINGW64 /i/POP (master)
$ git status
On branch master
Untracked files:
  (use "git add <file>..." to include in what will be committed)

        Assignments/
        Worksheets/
        git bash .txt
        git cmd.txt
        worksheet-object-oriented-programming-mconwa02/

nothing added to commit but untracked files present (use "git add" to track)
```

git log ~ shows a list of adds and commits, use q to quit

```
Michelle Conway@LAPTOP-SUCEBC4G MINGW64 /i/POP/Worksheets/Volume-of-a-Cone-Calculat
or-mconwa02 (master)
$ git log
commit 3d63a2979429f2839d72a9cbc1e6f1d27372c200 (HEAD -> master, origin/master, ori
gin/HEAD)
Author: Michelle Conway <32267382+mconwa02@users.noreply.github.com>
Date:   Sun Oct 22 16:56:31 2017 +0100

    Check and QC'd Volume of a cone

commit 74ef60e7c941d609c7f000b44a382049f8819ca3
Author: Keith Mannock <keith@dcs.bbk.ac.uk>
Date:   Wed Oct 11 13:58:43 2017 +0100

    Pytest added

commit c53493f07a19659bdb6e8992212733db5553f7b6
Author: Keith Mannock <keith@dcs.bbk.ac.uk>
Date:   Sun Sep 17 21:49:29 2017 +0100
```

git push origin master ~ push up all the changes and commits back to GitHub

```
Michelle Conway@LAPTOP-SUCEBC4G MINGW64 /i/POP/Worksheets/Volume-of-a-Cone-Calculat
or-mconwa02 (master)
$ git push origin master
fatal: HttpRequestException encountered.
   An error occurred while sending the request.
Username for 'https://github.com': mconwa02
Counting objects: 3, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 290 bytes | 1024 bytes/s, done.
Total 3 (delta 2), reused 0 (delta 0)
remote: Resolving deltas: 100% (2/2), completed with 2 local objects.
To https://github.com/BBK-DCSIS-PoP-I-2017-18/Volume-of-a-Cone-Calculator-mconwa02.
git
   3d63a29..7ed7852  master -> master
```

import random

Random module is used to generate random numbers like **randint(1,6)** generates whole integer values from 1 to and including 6.

random.random() returns a float N in the range 0 <= N < 1.0
random.randint(a, b**)** returns a integer value between a and b, inclusive
random.shuffle(seq **)** randomizes the sequence seq in place
random.choice(seq **)** returns a randomly chosen element of seq

```
>>> import random
>>> random.random()
0.814222452910484
>>> random.random()
0.2582303635035226
>>> a = [3, 5, 6, 7, 2, 5]
>>> random.shuffle(a)
>>> a
[5, 2, 7, 5, 6, 3]
>>> a
[5, 2, 7, 5, 6, 3]
>>> random.choice(a)
3
```

```
>>> from random import randint
>>> randint(1,6)
2
>>> randint(1,6)
5
>>> randint(1,6)
1
>>> randint(1,6)
6
>>> random.seed(3)
>>> random.random()
0.23796462709189137
>>> random.seed(3)
>>> random.random()
0.23796462709189137
```

random.seed(i **)** initializes the random number generator with the intger i, lets you use the same random sequence each time, like set.seed in R

From collections import Counter

Turns a sequence of values into a dictionary mapping keys to counts and gives most common values.

```
>>> from collections import Counter
>>> A = "word the is nothing betw tword if ther is happy and word"
>>> words = Counter(A)
>>> words
Counter({' ': 11, 't': 5, 'w': 4, 'o': 4, 'r': 4, 'd': 4, 'h': 4, 'i': 4, 'e
': 3, 'n': 3, 's': 2, 'a': 2, 'p': 2, 'g': 1, 'b': 1, 'f': 1, 'y': 1})
>>> words.most_common(5)
[(' ', 11), ('t', 5), ('w', 4), ('o', 4), ('r', 4)]
```

You can also use **randrange(1, 10)** uses the same principal as range function doesn't include last value 10 so only prints integers 1 to 9.

```
>>> from random import randrange
>>> randrange(1, 10)
5
>>> randrange(1, 10)
6
>>> randrange(1, 10)
5
>>> randrange(1, 10)
2
```

List ~ sometimes called arrays

A list is a sequence of elements numbered from 0 just like characters in a string and are mutable. Like string also does negative index, which means you start at the last element and go left.

Len() to get the number of elements in a list. To create a new list, the simplest is to enclose the elements in square brackets. The elements of a list don't have to be the same type.

```
>>> Primes = [2, 3, 5, 7, 11, 13]
>>> Primes[-1]
13
>>> len(Primes)
6
```

Unlike strings the elements are interactable in a list. In operator works on Boolean logic

```
>>> a = [2,4,3,b,7]
>>> 7 in a
True
>>> 9 in a
False
```

```
>>> Rainbow = ['Red', 'Orange', 'Yellow', 'Green', 'Blue', 'Indigo', 'Violet']
>>> print(Rainbow[0])
Red
>>> Rainbow[0] = 'red'
>>> for i in range(len(Rainbow)):
        print(Rainbow[i])


red
Orange
Yellow
Green
Blue
Indigo
Violet
```

Several ways to create a list. Can create an empty list using **= [ ] or list()** and add items to it using **append** method. **Extend** method takes a list as an argument and appends all the elements. Or create a dummy list and replace the elements.

```
>>> a = []
>>> a.append(55)
>>> print(a)
[55]
```

```
>>> a = ['d']*10
>>> a[0]= "T"
>>> a[1]='G'
>>> print(a)
['T', 'G', 'd', 'd', 'd', 'd', 'd', 'd', 'd', 'd']
```

```
>>> t1 = ['a', 'b', 'c']
>>> t2 = ['d', 'e']
>>> t1.extend(t2)
>>> t1
['a', 'b', 'c', 'd', 'e']
```

Sort() arranges the elements of the list from low to high

```
>>> t = ['d', 'c', 'e', 'b', 'a']
>>> t
['d', 'c', 'e', 'b', 'a']
>>> t.sort()
>>> t
['a', 'b', 'c', 'd', 'e']
```

Or if you don't want to mess up the list used sorted

```
>>> t = ['d','c', 'e', 'b','a']
>>> t
['d', 'c', 'e', 'b', 'a']
>>> x = sorted(t)
>>> x
['a', 'b', 'c', 'd', 'e']
>>> t
['d', 'c', 'e', 'b', 'a']
```

You can also define list using concatenation (addition) or repetition (multiplying)

```
>>> a = [1, 2, 3]
>>> b = [4, 5]
>>> c = a + b
>>> print(c)
[1, 2, 3, 4, 5]
```

```
>>> b = [4, 5]
>>> d = b*3
>>> print(d)
[4, 5, 4, 5, 4, 5]
>>> print([0,1]*3)
[0, 1, 0, 1, 0, 1]
```

You can use a **for loop** and print to print elements of a list in one line or one item per line.

Here the index i is changed then the element a[i] is displayed. While elem is not an index but rather a value of the variable that take values

```
>>> a = [1, 2, 3, 4, 5]
>>> for i in range(len(a)):
        print(a[i])


1
2
3
4
5
>>> for elem in a:
        print(elem, end = ' ')


1 2 3 4 5
```

```
>>> for i in range(len(a)):
        print(a[i], end = ' ')


1 2 3 4 5
```

```
>>> a = [1, 2, 3, 4, 5]
>>> for i in range(len(a)):
        print(a[i], end = '')


12345
```

Here is a clever way to **find** all numbers in a string. You loop through each element see if it is any digit 0-9 with find function returns -1 if symbol isn't in number string so numbers get **append** to empty list

```
>>> s = 'ab12c59p7dq'
>>> digits = []
>>> for symbol in s:
        if '1234567890'.find(symbol) != -1:
                digits.append(int(symbol))


>>> print(digits)
[1, 2, 5, 9, 7]
```

List slices work in the same way as string slices, it uses the indexs

```
>>> t = ['a', 'b', 'c', 'd', 'e']
>>> t[2:4]
['c', 'd']
>>> t[:]
['a', 'b', 'c', 'd', 'e']
>>> t[2:]
['c', 'd', 'e']
>>> t[:4]
['a', 'b', 'c', 'd']
```

Split() is a string methods which returns a list of strings after cutting the initial string by spaces. Watch out for **input()** it put everything in as strings so if you want a list of numbers you have to convert them using int().

```
>>> a = input().split()
3 5 2 5 3 5 3
>>> print(a)
['3', '5', '2', '5', '3', '5', '3']
>>> for i in range(len(a)):
        a[i] = int(a[i])

>>> print(a)
[3, 5, 2, 5, 3, 5, 3]
```

Or this can be shortened too….

```
>>> a = [int(s) for s in input().split()]
34 5 5 3 5 3 5 3 3 4
>>> print(a)
[34, 5, 5, 3, 5, 3, 5, 3, 3, 4]
```

Split can also use an optional parameter to separate items in a string like split('.')

```
>>> a = '192.168.0.1'.split('.')
>>> print(a)
['192', '168', '0', '1']
```

Join method displays a list of stings in one-line commands….

```
>>> a = ['red', 'green', 'blue']
>>> print(' '.join(a))
red green blue
>>> print(''.join(a))
redgreenblue
>>> print('***'.join(a))
red***green***blue
```

In joins if the list is numbers you have to convert to strings!

```
>>> a = [1, 2, 3]
>>> print(' '.join([str(s) for s in a]))
1 2 3
```

Substrings and sequences can be done on lists the same as strings. With negative three arguments giving opposite order.

Deleting elements, you can remove by the index with **pop** if you don't provide an index last element is deleted or delete an element using **del**.

```
>>> t = ['a', 'b', 'c']
>>> t.pop(1)
'b'
```

```
>>> t = ['a', 'b', 'c']
>>> t.pop()
'c'
```

```
>>> t = ['a', 'b', 'c']
>>> del t[2]
>>> t
['a', 'b']
```

You can also use **remove** if you know the element or index you want to remove.

```
>>> t = ['a', 'b', 'c']
>>> t.remove(t[2])
>>> t
['a', 'b']
```

```
>>> t = ['a', 'b', 'c']
>>> t.remove('a')
>>> t
['b', 'c']
```

To remove more than one element, you can use del with a slice index:
To convert from a string to a list of characters, you can use list

```
>>> t = ['a', 'b', 'c', 'd', 'e', 'f']
>>> del t[1:5]
>>> t
['a', 'f']
```

```
>>> s = 'spam'
>>> t = list(s)
>>> t
['s', 'p', 'a', 'm']
```

Lists unlike strings are **mutable** objects you can assign a list item to a new value. it is often useful to make a copy before performing operations that modify lists.

```
>>> t = [3,1,2]
>>> t2 = t[:]
>>> t2.sort()
>>> t
[3, 1, 2]
>>> t2
[1, 2, 3]
```

Please note that A[i] is not a slice but an item.

```
>>> A = [1, 2, 3, 4, 5, 6, 7]
>>> A[2:4] = [7, 8, 9]
>>> print(A)
[1, 2, 7, 8, 9, 5, 6, 7]
>>> A[::-2] = [10, 20, 30, 40]
>>> print(A)
[1, 40, 7, 30, 9, 20, 6, 10]
```

```
>>> a =[34, 5, 5, 3, 5, 3, 5, 3, 3, 4]
>>> a[2:5]
[5, 3, 5]
>>> a[5]
3
>>> a[0:8:2]
[34, 5, 5, 5]
>>> a[::-1]
[4, 3, 3, 5, 3, 5, 3, 5, 5, 34]
```

List Methods

sort	isupper	join	pop	remove	lower
append	islower	split	del	count	upper

List operations

A is b	Check if two lists are equal, return true or falase
x in A	Check if item in A, return true or false
x not in A	Check item not in A, return true or false
min(a)	Smallest element in list A
max(a)	Largest element in list A
A.index(x)	Index of first occurrence in list, none give exception ValueError
A.count(x)	The number of occurrences of element x in the list

A set is an immutable, unordered collection of values, containing no duplicates. A set can be set by naming all elements in brackets {} or as a string in set() or an empty set can be obtained by set() cannot use {} because that's a dictionary! Elements of a asset can be a number, string, tuple so mutable (changeable) data types cannot be elements of sets. So, lists can't be an element of a set!

For loop can be used in sets and len()

```
>>> set_A = {3,5,2,3}
>>> for elem in set_A:
        print(elem)


2
3
5
>>> len(set_A)
3
```

```
>>> set_B = set('eyegyebdhey')
>>> for elem in set_B:
        print(elem, end = ' ')

h g b y d e
>>> len(set_B)
6
```

To check is an element is in a set use in and to add elements to a set use add

```
>>> A = {1, 2, 3}
>>> print(1 in A, 4 not in A)
True True
>>> A.add(4)
>>> A
{1, 2, 3, 4}
```

There are two methods to remove elements from a set use discard and remove, both delete but it element not there remove throw an error discard doesn't. Pop removes the element at the top. Sets can be transformed to lists using list.

```
>>> B = {1, 2, 3, 4, 5}
>>> B.remove(5)
>>> B.discard(6)
>>> B.remove(6)
Traceback (most recent call last):
  File "<pyshell#30>", line 1, in <module>
    B.remove(6)
KeyError: 6
>>> B.pop()
1
>>> B
{2, 3, 4}
```

Can add elements to a set using add.

```
>>> B = {2,4,5,2,5,6,3,5,3,4,}
>>> B
{2, 3, 4, 5, 6}
>>> type(B)
<class 'set'>
>>> B.add(9)
>>> B
{2, 3, 4, 5, 6, 9}
```

A | B A.union(B)	Returns a set which is the union of sets A and B.
A |= B A.update(B)	Adds all elements of array B to the set A.
A & B A.intersection(B)	Returns a set which is the intersection of sets A and B.
A &= B A.intersection_update(B)	Leaves in the set A only items that belong to the set B.
A - B A.difference(B)	Returns the set difference of A and B (the elements included in A, but not included in B).
A -= B A.difference_update(B)	Removes all elements of B from the set A.
A ^ B A.symmetric_difference(B)	Returns the symmetric difference of sets A and B (the elements belonging to either A or B, but not to both sets simultaneously).
A ^= B A.symmetric_difference_update(B)	Writes in A the symmetric difference of sets A and B.
A <= B A.issubset(B)	Returns true if A is a subset of B.
A >= B A.issuperset(B)	Returns true if B is a subset of A.
A < B	Equivalent to A <= B and A != B
A > B	Equivalent to A >= B and A != B

A **dictionary** associates immutable **keys** with values, you can index into a dictionary like you would a list, but the “indices” are keys, not necessarily integers! Lists are not allowed be used a keys because they are mutable and keys need to be immutable for the hashing mapping to work.
Dictionaries have a method called items that returns a sequence of tuples, where each tuple is a key-value pair in no order. Likes use keys() for keys and values() for values

```
>>> d = {'a':0, 'b':1, 'c':2}
>>> t = d.items()
>>> t
dict_items([('a', 0), ('b', 1), ('c', 2)])
>>> for key, value in d.items():
        print(key, value)


a 0
b 1
c 2
```

```
>>> d = {'a':1, 'b':2, 'c':3}
>>> for key in d.keys():
        print(key)


a
b
c

>>> for val in d.values():
        print(val)


1
2
3
```

Using zip in dict gives an easy way to create a dictionary. An empty dictionary can be created with dict() Or use squiggly brackets {}

```
>>> d = dict(zip('abc', range(3)))
>>> d
{'a': 0, 'b': 1, 'c': 2}
```

The order of the items in a dictionary are unpredictable as numerical indies is not used. The in operator used to check if value in dictionary. Python uses hash table to search in a dictionary, so the time is always the same for a search unlike lists.

Dictionary can be used as counters

```
def histogram(s):
    d = dict()
    for c in s:
        if c not in d:
            d[c] = 1
        else:
            d[c] += 1
    return d
```

```
>>> histogram('shucnrivngcnf')
{'s': 1, 'h': 1, 'u': 1, 'c': 2, 'n': 3, 'r': 1, 'i': 1, 'v': 1, 'g': 1, 'f': 1}
```

To return key values in a dictionary like a list you use [] and index or dictionary you use get. We can provide a default value when key not in dictionary

```
>>> h = histogram('parrot')
>>> h.get('p', 3)
1
>>> h.get('p', 1)
1
>>> h.get('x', 3)
3
>>> h
{'p': 1, 'a': 1, 'r': 2, 'o': 1, 't': 1}
```

```
def histogram2(s):
    """using get to make this faster"""
    d = dict()
    for c in s:
        d[c] = d.get(c, 0) + 1
    return d
```

To sort a dictionary use sorted

```
>>> h = histogram('parrot')
>>> for key in sorted(h):
        print(key, h[key])


a 1
o 1
p 1
r 2
t 1
```

Reverse sort using reserve = TRUE

```
>>> words = Counter(A)
>>> words
Counter({' ': 11, 't': 5, 'w': 4, 'o': 4, 'r': 4, 'd': 4, 'h': 4, 'i': 4, 'e
': 3, 'n': 3, 's': 2, 'a': 2, 'p': 2, 'g': 1, 'b': 1, 'f': 1, 'y': 1})
>>> sorted(words, reverse = True)
['y', 'w', 't', 's', 'r', 'p', 'o', 'n', 'i', 'h', 'g', 'f', 'e', 'd', 'b',
'a', ' ']
```

Or use lambda to sort by values not keys
Reverse look ups are harder in dictionary could be multiple values as its only the keys that are distinct.

```
>>> sorted(words.items(), key = lambda x:1, reverse = True)
[('w', 4), ('o', 4), ('r', 4), ('d', 4), (' ', 11), ('t', 5), ('h', 4), ('e'
, 3), ('i', 4), ('s', 2), ('n', 3), ('g', 1), ('b', 1), ('f', 1), ('a', 2),
('p', 2), ('y', 1)]
>>> h = histogram('parrot')
>>> reverse(h, 3)
Traceback (most recent call last):
  File "<pyshell#12>", line 1, in <module>
    reverse(h, 3)
  File "C:\Users\Michelle Conway\Documents\MSC Y1 Principles of Programming\Snak
ify\Snakiy Lesson 11.py", line 14, in reverse
    raise Lookuperror('value not in dictioanry')
NameError: name 'Lookuperror' is not defined
>>> reverse(h, 2)
'r'
```

Hash function is one that take any kind of value and returns an integer. It works when everything is immutable. This make Fibonacci numbers faster than recursion

```
def fibonacci(n):
    if n in known:
        return known[n]

    res = fibonacci(n-1) + fibonacci(n-2)
    known[n] = res
    return res
```

```
>>> fibonacci(3)
2
>>> fibonacci(7)
13
```

To remove a key from a dictionary use del dict[key] or another way to remove is dict.pop(key) you can add second parameter to give when key not in dictionary

A **tuple** is immutable and an easy a way to collect a **small** number of related values into a single value, for convenience in handling. You can index into a tuple but can't change the values. Value can be any type, values in parenthesis is not a tuple. Use comma-separated list of values if just one value end with comma. Can also create a tuple with tuple()

```
>>> t = 'a', 4, 'b'
>>> type(t)
<class 'tuple'>
```

```
>>> t2 = 'm',
>>> type(t2)
<class 'tuple'>
```

```
>>> t3 = tuple()
>>> type(t3)
<class 'tuple'>
```

If the arguemnts in tuple is string, list or tuple the result is a tuple with the elements of the sequnece. Indexes and slices work on tuples.

```
>>> t4 = tuple('sillypops')
>>> t4
('s', 'i', 'l', 'l', 'y', 'p', 'o', 'p', 's')
>>> type(t4)
<class 'tuple'>
```

```
>>> t4[0]
's'
>>> t4[5:8]
('p', 'o', 'p')
```

Divmod takes two arguments and returns a tuple of two values, floor divider (aka quotient) and remainder

```
>>> divmod(5, 2)
(2, 1)
```

```
>>> 5 // 2
2
>>> 5 % 2
1
```

```
>>> quot, rem = divmod(7, 3)
>>> quot
2
>>> rem
1
```

Functions can take a variable number of arguments, parameters starting with * gather arguments into a tuple! Built in functions like max and min can take any number of arguments, but sum can't, a function called sumall to take any number of arguments and return their sum.

```
>>> max(4,3,5,6,4,6,4,77)
77
>>> min(3,2,3,4,3,0)
0
>>> sum(4,5,6,7)
Traceback (most recent call last):
  File "<pyshell#43>", line 1, in <module>
    sum(4,5,6,7)
TypeError: sum expected at most 2 arguments, got 4
```

```
def sumall(*t):
        tots = tuple(t)
        return sum(tots)

>>> sumall(4,3)
7
>>> sumall(3,4,5,6)
18
```

Zip is a built-in function that takes two or more arguments and returns a list of tuples where each tuple contains an element from each sequence. If sequences aren't same length the shortest length is taken.

```
>>> s = 'abcd'
>>> t = [0,1,2,3]
>>> r = 'x3y5'
>>> for pairs in zip(r,s,t):
        print(pairs)


('x', 'a', 0)
('3', 'b', 1)
('y', 'c', 2)
('5', 'd', 3)
```

Zip is more of an iterator unlike lists can't use index to select an element, so make zip a list!

```
>>> list(zip(r,s,t))
[('x', 'a', 0), ('3', 'b', 1), ('y', 'c', 2), ('5', 'd', 3)]
```

For Loop to traverse a list of tuples

```
>>> m = list(zip(r,s,t))
>>> m
[('x', 'a', 0), ('3', 'b', 1), ('y', 'c', 2), ('5', 'd', 3)]
>>> for a,b,c in m:
        print(a,b,c)


x a 0
3 b 1
y c 2
5 d 3
```

Unit Testing

When it fails it tells you what piece of code needs to be fixed. It tests each individual piece of code.

Integration testing

When it fails it tells you that the application is not doing what the customer expects it to do. It puts several units together that interact and tests to make sure that integrating these units together doesn't introduce any errors.

Regression test

When it fails, it tells you that the application no longer behaves the way it used to. After integration testing you should run unit tests again this is regression testing so insure any further changes have not broken the unit tests.

Acceptance test

When business function conduct acceptance tests to ensure functionality meets their requirements

Import statements are needed for the file you are testing, keep program and test file separate eg import myprog.py

If import just the programme eg myprog.py must reference the program when calling any of its functions. Eg myprog.myfun(args)

Or called everthing in the import, from myprog import *

Structure

Looks like the below, always begin each test with test_ and parameter self. Inside test normal python but no input or output. Make code free of I/0

```
import unittest
from myprog import *

class nameOfClass(unittest.TestCase):

    def test_mathod1:
            blah
            blah

    def test_mathod2:
            blah
            blah

unittest.main()
```

Using self to test

```
self.assertTrue()
self.assertFalse()
self.assertEqual()
```

Exampe of using self looks like

```
def test_exmaple(self):
    n = 0
    for i in range(10):
        n = n + 1
        self.assertEqual(10, n)
```

Remember floating point numbers not accurate so don't use assert use Almost Assert

```
def test_exmaple(self):
    n = 0
    for i in range(10):
        n = n + 0.1
        self.assertAlmostEqual(1.0, n)
```

More test methods

```
aseertEqual(a, b)          assertEqual(a, b, message)
assertNotEqual(a , b)      assertNotEqual(a, b , message )
assertFalse(x)             assertFalse(x , message )
assertIsNotNone(x)         assertIsNotNone(x , message )
assertIsInstance(a, b)     assertIsInstance(a , b , message )
```

White hat hacker triy to find security flaws in order to get company to fix them, these are the good guys.

Black hat hackers try to find the security flaws in order to exploit them, these are the bag guys.

Garden path testing solving simple and most common cases but should also look for edge cases, the extreme and unexpected

TDD Test Driven Development

- Write simple test for code you plan to write
- Write code
- Run test, debug until it works
- Clean up (refactor) the code, make sure it still works
- If code doesn't do everything you want to do, write test for next feature and go to step 2.

Pytest

- Write simple test for code you plan to write

Write

TDD benefits

- Tests first make functions small and single purpose
- Avoids long and painful debugging sessions
- Promotes steady step-by-step approach

Refactoring means changing code to make it faster without changing what it does.

- Common ways, change variable name or function to better meaning

- Simplify arithmetic expressions by giving names to parts and using name in expression

Special code

- Programs typical have a method named main which is the starting point for everything in the program.
- Started automatically when loaded but putting on the last line main()
- The following is magic code placed at the end of the program, will call main method to start the program if and only if you run the code from the file

```
if __name__ == '__main__':
    main()
```

- Doesn't work if you run test from separate file, the above code does not call the main method to start the program running.

Two -Dimensional Lists ~ sometimes called matrix or 2D arrays

In Python presented as a list of lists, you iterate through the array by row number then by elements in row.

```
A = [[2, 2, 5], [3, 6], [7, 9, 2, 3]]
for row in A:
    for elem in row:
        print(elem, end = ' ')
```

```
2 2 5 3 6 7 9 2 3
```

Or to output in a single line:

```
for row in A:
    print(' '.join([str(elem) for elem in row]))
```

```
2 2 5
3 6
7 9 2 3
```

Creating nested lists need to be careful, creating an array of n rows and m columns using

```
A = [[0]*m]*n
```

Will just reference a list of m zeros and not a list. To avoid this use

```
m = 7
n = 3
A = [0]*n
for i in range(n):
    A[i]=[0]*m
```

```
n = 3
m = 4
A = []
for i in range(n):
    a.append([0]*m)
```

```
n = 3
m = 4
A = [[0]*m for i in range(n)]
```

Using 2D lists to multiple matrix, notice the print out statement used join twice once with space and new line

```
def two_matrix(m, n, r):
    A = [[ 2 for i in range(n)] for i in range(m)]
    B = [[ 3 for i in range(r)] for i in range(n)]
    C = [[ 0 for i in range(r)] for i in range(m)]
    for i in range(m):
        for j in range(r):
            for k in range(n):
                C[i][j] += A[i][k]*B[k][j]
    print('\n'.join([' '.join([str(s) for s in row]) for row in C ]))
```

Recursion

When a function calls itself is recursive and the process of existing it is called recursion. Stacking happens during recursion, the bottom of the stack is called base case. If a recursion never reaches a base case , it goes on making recursive calls forever and programme never terminates. This is called infinite recursion.

Recursion examples:

- Reverse a sequence
- Count down function
- Factorial function
- Fibonacci series
- Exponent function

Reverse a sequence

```
def reverse():
    x = int(input())
    if x != 0:
        reverse()
    print(x)
```

```
>>> reverse()
4
5
3
4
6
30
0
0
30
6
4
3
5
4
>>>
```

Count down function

```
def print_n(s, n):
    if n <= 0:
        return
    print(s)
    print_n(s, n-1)
```

```
>>> print_n(4, 2)
4
4
>>> print_n(16, 4)
16
16
16
16
```

Factorial function

```
def factorial(n):
    if n == 1:
        return 1
    else:
        return n*factorial(n-1)
```

```
>>> factorial(3)
6
>>> factorial(5)
120
>>> factorial(7)
5040
```

Fibonacci series

```
def fib(n):
    if n == 1:
        return 1
    elif n == 2:
        return 1
    else:
        return fib(n-1) + fib(n-2)
```

```
>>> fib(3)
2
>>> fib(5)
5
>>> fib(1)
1
```

```
>>> fib(2)
1
>>> fib(7)
13
```

Exponent function

```
def exponent(a, n):
    if n == 0:
        return 1
    else:
        return a * exponent(a, n - 1)
```

```
>>> exponent(5, 3)
125
>>> exponent(4, 3)
64
```

```
>>> exponent(2, 2)
4
>>> exponent(2, 3)
8
>>>
```

Iterables

An iterator is something that return a value one at a time. Sets, tuples, dictionaries, ranges and files can be iterated, for uses iterators. Range works in for but not where a list is required.

Yeild

Yiled is like return but you can come back to the function and continue where you yielded and the state of local variables is retained

Object Oriented Programming OOP

Classes and Objects

Class

Every object has attributes which are object variables

Every object has abilities which are object functions

Constructor called to set up or initialise an object inside a class

self allows an object to refer to itself inside of the class

```
class Animal:
    # None signifies the lack of a value
    # You can make a variable private by starting it with __
    __name = None
    __height = None
    __weight = None
    __sound = None
```

```
    def __init__(self, name, height, weight, sound):
        self.__name = name
        self.__height = height
        self.__weight = weight
        self.__sound = sound
```

```
# How to create a Animal object
cat = Animal('Whiskers', 33, 10, 'Meow')
```

You can't access this value directly because it is private
#print(cat.__name)

Self means allows an object to revert itself inside the class

Encapsulation setting private names

Functions and attributes inside a class

Inheritance you inherit from another class you get all variables and functions/methods from that class

Can overwrite constructor

Can use super to let original class variable be kept in constructor

```
class Dog(Animal):
    __owner = None

    def __init__(self, name, height, weight, sound, owner):
        self.__owner = owner
        self.__animal_type = None

        # How to call the super class constructor
        super(Dog, self).__init__(name, height, weight, sound)
```

Method overloading, preforming different tasks based on attributes

Polymorphism allows use to refer to objects as their super class and correct functions are called automatically

Files

```
# Overwrite or create a file for writing
test_file = open("test.txt", "wb")

# Get the file mode used
print(test_file.mode)

# Get the files name
print(test_file.name)

# Write text to a file with a newline
test_file.write(bytes("Write me to the file\n", 'UTF-8'))

# Close the file
test_file.close()
```

```
# Opens a file for reading and writing
test_file = open("test.txt", "r+")

# Read text from the file
text_in_file = test_file.read()

print(text_in_file)

# Delete the file
os.remove("test.txt")
```

```
# Opens a file for reading and writing
test_file = open("test.txt", "r+")

# Read text from the file
text_in_file = test_file.read()

print(text_in_file)

# Delete the file
os.remove("test.txt")
```

```
# Overwrite or create a file for writing
test_file = open("test.txt", "wb")

# Get the file mode used
print(test_file.mode)

# Get the files name
print(test_file.name)

# Write text to a file with a newline
test_file.write(bytes("Write me to the file\n", 'UTF-8'))

# Close the file
test_file.close()
```

- **Recursion is** a tool a programmer can use to invoke a function call on itself.
- **Side effect** is when a procedure/function changes a variable from outside its scope. Like global variable not in a function
- **dynamic programming** (also known as **dynamic** optimization) is a method for solving a complex problem by breaking it down into a collection of simpler subproblems, solving each of those subproblems just once, and storing their solutions.
- **Magic numbers** are any number in code that isn't immediately obvious to someone with very little knowledge that it is a number. Gravity = 8.7 or len(matrix) equals 5
- **Encapsulation** is one of the fundamental concepts in object-oriented programming (OOP). It describes the idea of bundling data and methods that work on that data within one unit, e.g., a class
- **refactoring** is the process of restructuring existing codewithout changing its external behaviour.
- **Style Guide for Python Code**
- As code is read much more often than it is written.
- The purpose of style guide intended to improve the readability of code and its intention is to keep it consistent across the wide spectrum of Python code
- REPL Read Eval Print Loop
- TDD Test Driven Development
- Python 2 uses ASCII Python 3 uses Unicode
- Unit testing, the unit is the individual unit of source code in the program that is being tested

don't repeat yourself (**DRY**)

- It is not a good idea to “copy and paste" code, or otherwise duplicate code is considered undesirable. If the code has a vulnerability this will continue to

exist in the copied code. Example if you just use one function and reference it multiple time and the function is incorrect you only have to do one fix and edit the function and not a serval fixes in multiple places

- It is not a good idea to have the same information represented in one way as it is important to keep it consistent to avoid redundancy. List versus sets

OBB principles

- Inheritance
- Encapsulation

Pair programming is an agile software development technique in which two programmers work together at one workstation. One, the *driver*, writes code while the observer or *navigator*, reviews each line of code as it is typed in. The two programmers switch roles frequently.