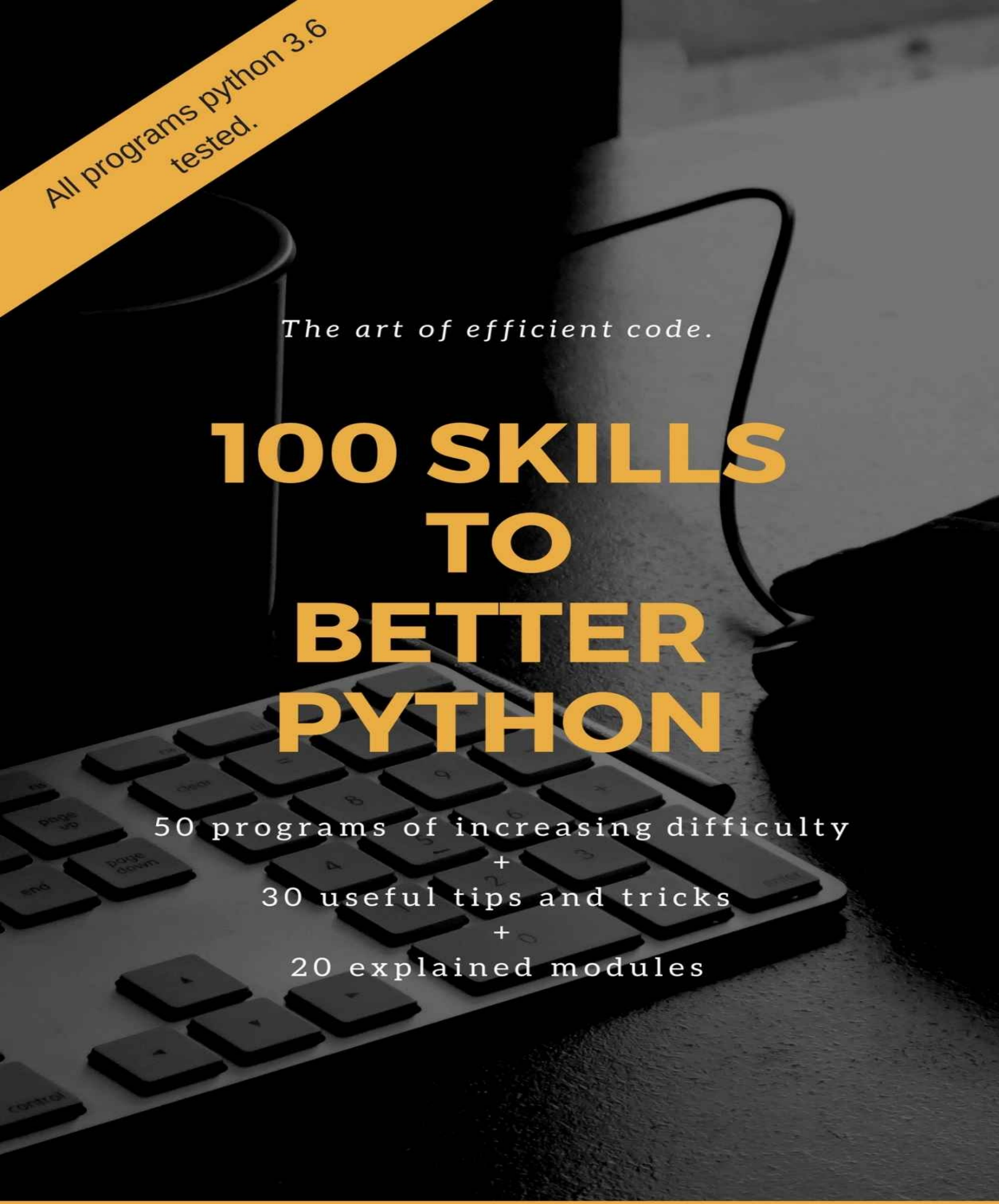

The art of efficient code.

100 SKILLS TO BETTER PYTHON

50 programs of increasing difficulty
+
30 useful tips and tricks
+
20 explained modules

ADITYA PRASANNA

Preface:

As a beginner in programming, often one way or another, we stumble across Python as a go-to multifunctional powerhouse. Like the other languages, we also see that there is no prescribed course that will guarantee you to be a successful Python programmer. Our aim was to build a workbook that would be conducive to a programmer's journey of learning. In this book you will find everything that two experienced Python professionals feel is essential for a well-rounded, in-depth perception of the language. Although not everything can be covered no matter the given time, we attempt to cover a wide variety of Python essentials in this workbook. You will find that the exemplary programs make best use of numerous Python functions and methodologies such that topics are not repeated. Although you will find a brief explanation of the functions, feel free to utilize the wide range of resources offered throughout the internet keeping in mind there is never only one way to solve a problem. This book will not cover the basics such as the syntax, structure, etc. as you can easily find clear explanations everywhere. Welcome to the world of Python. Let's get started.

Prerequisites:

-Python 3.6 downloaded and installed on your local system
-A compiler (suggested use: eclipse, jupyter notebook)
-A text editor (suggested use: sublime text, atom)
-Knowledge of the syntax and structure of Python 3

Introduction:

All the programs and tips mentioned in this book are tested in Python 3.6. In the program section, we introduce three levels of difficulty: Beginner (20), Intermediate (20), Advance (20). These will cover the most essential topics in the world of Python 3. The next section will cover tips and tricks to make your Python code shorter, faster and more efficient. This will be followed by a comprehensive explanation of essential modules and packages offered by either Python or 3rd party open source organizations. For your own betterment, try to understand, type and implement all that you learn instead of reading through the examples.

Table of Contents:

Beginner:

These are beginner concepts you should have at the tip of your fingers. Good luck.

1. With a given number n, write a program to generate a dictionary that contains (i, i*i) such that i is an number between 1 and n (both included). and then the program should print the dictionary.

```
n=int(input("Enter a number:"))
d=dict()
for i in range(1,n+1):
    d[i]=i*i
print(d)
```

2. Write a program which accepts a sequence of comma-separated numbers from console/user and generates a list and a tuple which contains every number.

```
values=input("Enter comma-separated numbers:")
l=values.split(",")
t=tuple(l)
print(l)
print(t)
```

3. Write a program which will find all the numbers which are divisible by 7 but are not a multiple of 5, between 1000 and 1500 (both included). The numbers obtained should be printed in a comma-separated sequence on a single line.

```
l=[]
for i in range(1000, 1500):
    if (i%7==0) and (i%5!=0):
        l.append(str(i))
t=','.join(l)
print(t)
```

4. Define a function which can compute the sum of two numbers.

```
def printValue(s1,s2):
    print(int(s1)+int(s2))
printValue("3","4")
```

5. Define a function that can receive two integral numbers in string form and compute their sum and then print it in console.

```
def SumFunction(number1, number2):
    return number1+number2
print(SumFunction(1,2))
```

6. Write a program which can compute the given factorial of a number.

```
def fact(x):
    if x == 0 or x == 1:
        return 1
    return x * fact(x - 1)

x=int(input())
print(fact(x))
```

7. Use list comprehension to square each odd number in a list. The list is input by a sequence of comma-separated numbers.

```
value=input()
numbers=[str(int(x)**2) for x in value.split(",") if (int(x)%2!=0)]
print(','.join(numbers))
```

8. Write a program to roll a dice and get a random output (1-6).

```
import random   #random module is imported here
min = 1
max = 6
roll_again = "yes"

while roll_again == "yes" or roll_again == "y":
    print("Rolling the dices...")
    print("The values are....")
    print(random.randint(min, max))
    print(random.randint(min, max))
    roll_again = input("Roll the dices again?")
```

9. Define a function which can generate a dictionary where the keys are numbers between 1 and 20 (both included) and the values are square of keys. The function should just print the values only.

```
def printDict():
    d=dict()
    for i in range(1,21):
        d[i]=i**2
    for (k,v) in d.items():
        print(v)
printDict()
```

10. Define a class which has at least two methods: getstring: to get a string from user. printstring: to print the string in upper case. Include a test function to test the class methods.

```
class InputOutString(object):
    def __init__(self):
        self.s = ""
    def getString(self):
        self.s = input()
        t = self.s
    def printString(self):
        l = t.upper()
        print(l)

strObj = InputOutString()
strObj.getString()
strObj.printString()
```

11. Define a class, which have a class parameter and have a same instance parameter.

```
class Person:
    # Define the class parameter "name"
    name = "Person"
        def __init__(self, name = None):
    # self.name is the instance parameter
        self.name = name

jon = Person("Jon")
print("%s name is %s" % (Person.name, jon.name))
anderson = Person()
anderson.name = "Anderson"
print("%s name is %s" % (Person.name, anderson.name))
```

12. Write a program that accepts a sentence and calculates the number of upper and lower case letters.

```
s = input("Enter a sentence:")
d={"UPPER CASE":0, "LOWER CASE":0}
for c in s:
    if c.isupper():
        d["UPPER CASE"]+=1
    elif c.islower():
        d["LOWER CASE"]+=1
    else:
        pass
print("UPPER CASE", d["UPPER CASE"])
print("LOWER CASE", d["LOWER CASE"])
```

13. Write a program to display the fibonacci series up to the nth term where nth term is given by the user.

```
nterms = int(input("Enter value of n:"))
# first two terms
n1 = 0
n2 = 1
count = 0
# check if the number of terms is valid
if nterms <= 0:
   print("Please enter a positive integer")
elif nterms == 1:
   print("Fibonacci sequence upto",nterms,":")
   print(n1)
else:
   print("Fibonacci sequence upto",nterms,":")
   while count < nterms:
       print(n1,end=' , ')
       nth = n1 + n2
       # update values
       n1 = n2
       n2 = nth
       count += 1
```

14. Define a class named American and its subclass NewYorker.

```
class American(object):
    pass

class NewYorker(American):
    pass

anAmerican = American()
aNewYorker = NewYorker()
print(anAmerican)
print(aNewYorker)
```

15. Define a class named Circle which can be constructed by a radius. The Circle class has a method which can compute the area.

```
class Circle(object):
    def __init__(self, r):
        self.radius = r

    def area(self):
        return (self.radius**2)*3.14

aCircle = Circle(2)
print(aCircle.area())
```

16. Write a program using generator to print the even numbers between 0 and n in comma separated form while n is input by console.

```
def EvenGenerator(n):
    i=0
    while i<=n:
        if i%2==0:
            yield i
        i+=1

n=int(input())
values = []
for i in EvenGenerator(n):
    values.append(str(i))
print(",".join(values))
```

17. Write statements using assert expression to verify that every number in the list [2,4,6,8] is even.

```
li = [2,4,6,8]
for i in li:
    assert i%2==0
#check by appending list with an odd number to get assertion error
```

18. Write a program to compress and decompress the string "Hello world! Python is great!".

```python
import zlib
s = "Hello world! Python is great!"
t = zlib.compress(s.encode("utf-8"))
print(t)
print(zlib.decompress(t))
```

19. Define three individual functions to implement the filter, map and reduce functions. Experiment on them as you like.

```python
import functools

def f(x): return x % 2 != 0 and x % 3 != 0
list(filter(f, range(2, 25)))

def cube(x): return x*x*x
list(map(cube, range(1, 11)))

def add(x,y): return x+y
functools.reduce(add, range(1, 11))
```

20. Create a list of integers. Using filter and lambda functions find the integers that are multiples of 3. Using map and lambda functions multiply all integers of the list by 2 and add 15. Use the reduce function from functools module to simply add all integers.

```
import functools
foo = [2, 18, 9, 22, 17, 24, 8, 12, 27]

print(list(filter(lambda x: x % 3 == 0, foo)) )
print(list(map(lambda x: x * 2 + 15, foo)))
print(functools.reduce(lambda x, y: x + y, foo))
```

Intermediate:

The following programs are relatively tougher than the ones in the previous section. In this sections, many of the important sorting and searching algorithms have been included.

21. Write a program to compute 1/2+2/3+3/4+...+n/n+1 with a given n input by console (n>0).

```
n=int(input())
sum=0.0
for i in range(1,n+1):
    sum += float(float(i)/(i+1))
print(sum)
```

22. With a given list [12,24,35,24,88,120,155,88,120,155], write a program to print this list after removing all duplicate values with original order reserved.

```
def removeDuplicate( li ):
    newli=[]
    seen = set()
    for item in li:
        if item not in seen:
            seen.add( item )
            newli.append(item)
    return newli

li=[12,24,35,24,88,120,155,88,120,155]
print(removeDuplicate(li))
```

23. In a given sentence, find all the adverbs and their positions using the re module.

```
import re
text = "Clearly, he felt she was inexcusably wrong"
for m in re.finditer(r"\w+ly", text):
    print('%d-%d: %s' % (m.start(), m.end(), m.group(0)))
```

24. Using the re module, find a way to remove anything between parenthesis in a given string.

```
import re
items = ["techotd(.com)", "theverge(.com)", "edx(.org)", "github(.com)", "stackoverflow(.com)"]
for item in items:
    print(re.sub(r" ?\([^)]+\)", "", item))
```

25. Open a text file and find the longest word in the text file and find the length.

```
def longest_word(filename):
    with open(filename, 'r') as infile:
            words = infile.read().split()
    max_len = len(max(words, key=len))
    return [word for word in words if len(word) == max_len]

print(longest_word('test.txt'))  #replace test with the name of your file
```

26. Open a text file and find out how many lines are in the text file.

```
def file_lengthy(fname):
        with open(fname) as f:
                for i, l in enumerate(f):
                        pass
        return i + 1
print("Number of lines in the file: ",file_lengthy("test.txt"))
#replace test with the name of your file
```

27. Using the NumPy module, create an array of floating point values, square and find absolute value of all elements.

```
import numpy as np
x = np.arange(7) #creates an array with range(7)

print("Original array")
print(x)
print("First array elements raised to powers from second array, element-wise:")
print(np.power(x, 3))
print("Element-wise absolute value:")
print(np.absolute(x))
```

28. Use the numpy module to compute the trigonometric sine, cosine and tangent array of angles given in degrees.

```
import numpy as np

print("sine: array of angles given in degrees")
print(np.sin(np.array((0., 30., 45., 60., 90.)) * np.pi / 180.))

print("cosine: array of angles given in degrees")
print(np.cos(np.array((0., 30., 45., 60., 90.)) * np.pi / 180.))

print("tangent: array of angles given in degrees")
print(np.tan(np.array((0., 30., 45., 60., 90.)) * np.pi / 180.))
```

29. Develop a program to multiply two matrices. First matrix of order 3x3 and second matrix of order 3x4.

```
# 3x3 matrix
X = [[12,7,3],
    [4 ,5,6],
    [7 ,8,9]]
# 3x4 matrix
Y = [[5,8,1,2],
    [6,7,3,0],
    [4,5,9,1]]
# result is 3x4
result = [[0,0,0,0],
          [0,0,0,0],
          [0,0,0,0]]

# iterate through rows of X
for i in range(len(X)):
   # iterate through columns of Y
   for j in range(len(Y[0])):
       # iterate through rows of Y
       for k in range(len(Y)):
           result[i][j] += X[i][k] * Y[k][j]

for r in result:
   print(r)
```

30 Design a program to create a diamond pattern using the asterisk symbol by taking the side length as input from user.

```
side = int(input("Please input side length of diamond: "))

for x in list(range(side)) + list(reversed(range(side-1))):
    print('{: <{w1}}{:*<{w2}}'.format('', '', w1=side-x-1, w2=x*2+1))
```

31. Develop a simple encryption and decryption program by shifting a character 2 ASCII values down for encryption and 2 ASCII values back up for decryption.

```
result=''
message=''
choice=''

while choice!=0:
    choice = input("\nDo you want to encrypt or decrypt the message?\
    nEnter 1 to encrypt, 2 to decrypt and 0 to exit the program.");

    if choice =='1':
        message=input("\nEnter message for encryption: ")
        for i in range(0, len(message)):
            result = result +chr(ord(message[i]) - 2)
        print(result + '\n\n')
        result = ''

    elif choice == '2':
        message = input("\nEnter the message to decrypt: ")
        for i in range(0, len(message)):
            result = result + chr(ord(message[i]) +2)

        print(result + '\n\n')
        result=''

    elif choice !='0':
        print("You have entered an invalid input!. Please try again. \n\n")
```

Note: The remaining programs are basic algorithm implementations that are crucial to have a grip on. There are numerous ways of implementation so the need to understand is greater than the need to memorize the steps.

32. Develop a function to implement **Binary Search**.

```python
import math
def bin_search(li, element):
    bottom = 0
    top = len(li)-1
    index = -1
    while top>=bottom and index==-1:
        mid = int(math.floor((top+bottom)/2.0))
        if li[mid]==element:
            index = mid
        elif li[mid]>element:
            top = mid-1
        else:
            bottom = mid+1

    return index

li=[2,5,7,9,11,17,222]
print(bin_search(li,11))
print(bin_search(li,12))
```

33. Write a function to implement **Linear/Sequential Search**.

```
def Sequential_Search(dlist, item):
    pos = 0
    found = False
    while pos < len(dlist) and not found:
        if dlist[pos] == item:
            found = True
        else:
            pos = pos + 1
    return found, pos

print(Sequential_Search([11,23,58,31,56,77,43,12,65,19],31))
```

34. Write a function to implement **Bubble Sort**.

```
def bubbleSort(nlist):
    for passnum in range(len(nlist)-1,0,-1):
        for i in range(passnum):
            if nlist[i]>nlist[i+1]:
                temp = nlist[i]
                nlist[i] = nlist[i+1]
                nlist[i+1] = temp

nlist = [14,46,43,27,57,41,45,21,70]
bubbleSort(nlist)
print(nlist)
```

35. Write a function to implement **Selection Sort**.

```
def selectionSort(nlist):
    for fillslot in range(len(nlist)-1,0,-1):
        maxpos=0
        for location in range(1,fillslot+1):
            if nlist[location]>nlist[maxpos]:
                maxpos = location

        temp = nlist[fillslot]
        nlist[fillslot] = nlist[maxpos]
        nlist[maxpos] = temp

nlist = [14,46,43,27,57,41,45,21,70]
selectionSort(nlist)
print(nlist)
```

36. Develop a function to implement **Insertion Sort**.

```
def insertionSort(nlist):
    for index in range(1,len(nlist)):

      currentvalue = nlist[index]
      position = index

      while position>0 and nlist[position-1]>currentvalue:
          nlist[position]=nlist[position-1]
          position = position-1

      nlist[position]=currentvalue

nlist = [14,46,43,27,57,41,45,21,70]
insertionSort(nlist)
print(nlist)
```

37. Develop a function to implement **Shell Sort**.

```
def shellSort(alist):
    sublistcount = len(alist)//2
    while sublistcount > 0:
      for start_position in range(sublistcount):
        gap_InsertionSort(alist, start_position, sublistcount)

      print("After increments of size",sublistcount, "The list is",nlist)
      sublistcount = sublistcount // 2
def gap_InsertionSort(nlist,start,gap):
    for i in range(start+gap,len(nlist),gap):

        current_value = nlist[i]
        position = i

        while position>=gap and nlist[position-gap]>current_value:
            nlist[position]=nlist[position-gap]
            position = position-gap
        nlist[position]=current_value

nlist = [14,46,43,27,57,41,45,21,70]
shellSort(nlist)
print(nlist)
```

38. Develop a function to implement **Quick Sort**.

```
def quickSort(data_list):
    quickSortHlp(data_list,0,len(data_list)-1)

def quickSortHlp(data_list,first,last):
    if first < last:

        splitpoint = partition(data_list,first,last)

        quickSortHlp(data_list,first,splitpoint-1)
        quickSortHlp(data_list,splitpoint+1,last)

def partition(data_list,first,last):
    pivotvalue = data_list[first]

    leftmark = first+1
    rightmark = last

    done = False
    while not done:

        while leftmark <= rightmark and data_list[leftmark] <= pivotvalue:
            leftmark = leftmark + 1

        while data_list[rightmark] >= pivotvalue and rightmark >= leftmark:
            rightmark = rightmark -1

        if rightmark < leftmark:
            done = True
        else:
            temp = data_list[leftmark]
            data_list[leftmark] = data_list[rightmark]
            data_list[rightmark] = temp

    temp = data_list[first]
    data_list[first] = data_list[rightmark]
    data_list[rightmark] = temp
    return rightmark

data_list = [54,26,93,17,77,31,44,55,20]
quickSort(data_list)
print(data_list)
```

39. Develop a function to implement **Merge Sort**.

```
def mergeSort(nlist):
    print("Splitting ",nlist)
    if len(nlist)>1:
        mid = len(nlist)//2
        lefthalf = nlist[:mid]
        righthalf = nlist[mid:]

        mergeSort(lefthalf)
        mergeSort(righthalf)
        i=j=k=0
        while i < len(lefthalf) and j < len(righthalf):
            if lefthalf[i] < righthalf[j]:
                nlist[k]=lefthalf[i]
                i=i+1
            else:
                nlist[k]=righthalf[j]
                j=j+1
            k=k+1

        while i < len(lefthalf):
            nlist[k]=lefthalf[i]
            i=i+1
            k=k+1

        while j < len(righthalf):
            nlist[k]=righthalf[j]
            j=j+1
            k=k+1
    print("Merging ",nlist)

nlist = [14,46,43,27,57,41,45,21,70]
mergeSort(nlist)
print(nlist)
```

40. Develop a function to implement **Counting Sort**.

```
def counting_sort(array1, max_val):
    m = max_val + 1
    count = [0] * m
    for a in array1:
    # count occurences
        count[a] += 1
    i = 0
    for a in range(m):
        for c in range(count[a]):
            array1[i] = a
            i += 1
    return array1
print(counting_sort( [1, 2, 7, 3, 2, 1, 4, 2, 3, 2, 1], 7 ))
```

Advanced:

The following programs are implementations of Non-primitive data structures. The primitive data structures in python as you have already seen are lists, tuples, dictionaries, etc. As an advanced python programmer, it is important to have knowledge of data structures, their working and their implementation. You will be required to use these data structures along with the algorithms in the previous section build effective programs. These data structures a basic implementations, which may not give you the desired output. You may have to tweak and experiment with your code to get what you require.

41. Implement a basic **Stack**.

```
class Stack:
    def __init__(self):
        self.items = []

    def isEmpty(self):
        return self.items == []  #checks if Stack is empty

    def push(self, item):
        self.items.append(item)  #inserts element into Stack

    def pop(self):
        return self.items.pop()  #removes element from Stack and returns it

    def peek(self):
        return self.items[len(self.items)-1]  #returns the value of top element

    def size(self):
        return len(self.items)  #returns size of stack
```

42. Implement a basic **Queue**.

```
class Queue:
    def __init__(self):
        self.items = []

    def isEmpty(self):
        return self.items == []  #checks if queue is empty

    def enqueue(self, item):
        self.items.insert(0,item)  #inserts element into queue

    def dequeue(self):
        return self.items.pop()  #removes and returns the value of last element in queue

    def size(self):
        return len(self.items)  #returns size of queue
```

43. Implement a basic **Deque**.

```python
class Deque:
    def __init__(self):
        self.items = []

    def isEmpty(self):
        return self.items == []  #checks if deque is empty

    def addFront(self, item):
        self.items.append(item)  #inserts element to front of queue

    def addRear(self, item):
        self.items.insert(0,item)  #inserts element to rear of queue

    def removeFront(self):
        return self.items.pop()  #removes element and returns value of last element

    def removeRear(self):
        return self.items.pop(0)  #removes element and returns value of first element

    def size(self):
        return len(self.items)  #returns size of queue
```

44. Implement a **Linked List**.

```
        return None                #searches for element in LinkedList

    def remove( self, p ) :
        tmp = p.prev
        p.prev.next = p.next
        p.prev = tmp                #removes element form LinkedList

    def __str__( self ) :
        s = ""
        p = self.head
        if p != None :
            while p.next != None :
                s += p.data
                p = p.next
            s += p.data
        return s
```

```
class Node :
    def __init__( self, data ) :
        self.data = data
        self.next = None
        self.prev = None  #class is called to create a new node

class LinkedList :
    def __init__( self ) :
        self.head = None

    def add( self, data ) :
        node = Node( data )
        if self.head == None :
            self.head = node
        else :
            node.next = self.head
            node.next.prev = node
            self.head = node        #node class is called to add new node

    def search( self, k ) :
        p = self.head
        if p != None :
            while p.next != None :
                if ( p.data == k ) :
                    return p
                p = p.next
            if ( p.data == k ) :
                return p
```

45. Implement a **Doubly Linked List**.

```
class Node():
    def __init__(self, next_node=None, previous_node=None, data=None):
        self.next_node = next_node
        self.previous_node = previous_node
        self.data = data

class LinkedList():
    def __init__(self, node):
        assert isinstance(node, Node)
        self.first_node = node
        self.last_node = node

    def push(self, node):
        '''Pushes the node <node> at the "front" of the ll
        '''
        node.next_node = self.first_node
        node.previous_node = None
        self.first_node.previous_node = node
        self.first_node = node

    def pop(self):
        '''Pops the last node out of the list'''
        old_last_node = self.last_node
        to_be_last = self.last_node.previous_node
        to_be_last.next_node = None
```

```
        old_last_node.previous_node = None
        # Set the last node to the "to_be_last"
        self.previous_node = to_be_last
        return old_last_node

    def remove(self, node):
        '''Removes and returns node, and connects the previous and next
        nicely
        '''
        next_node = node.next_node
        previous_node = node.previous_node

        previous_node.next_node = next_node
        next_node.previous_node = previous_node
        # Make it "free"
        node.next_node = node.previous_node = None
        return node

    def __str__(self):
        next_node = self.first_node
        s = ""
        while next_node:
            s += "--({:0>2d})--\n".format(next_node.data)
            next_node = next_node.next_node
        return s
```

```
        return

node1 = Node(data=1)
linked_list = LinkedList(node1)

for i in range(10):
    if i == 5:
        node5 = Node(data=5)
        linked_list.push(node5)
    else:
        linked_list.push(Node(data=i))

print(linked_list)

print("popping")
print(linked_list.pop().data)

print("\n\n")
print(linked_list)

print("\n\n")
linked_list.push(Node(data=10))

print("\n\n")
print(linked_list)

linked_list.remove(node5)

print("\n\n")
print(linked_list)
```

46. Implement a **Binary Tree**.

```
class BinaryTree():

    def __init__(self,rootid):
      self.left = None
      self.right = None
      self.rootid = rootid

    def getLeftChild(self):
        return self.left
    def getRightChild(self):
        return self.right
    def setNodeValue(self,value):
        self.rootid = value
    def getNodeValue(self):
        return self.rootid

    def insertRight(self,newNode):
        if self.right == None:
            self.right = BinaryTree(newNode)
        else:
            tree = BinaryTree(newNode)
            tree.right = self.right
            self.right = tree

```

Note: The following programs aren’t implementations of data structures. They are programs at a sligh

```
    def insertLeft(self,newNode):
        if self.left == None:
            self.left = BinaryTree(newNode)
        else:
            tree = BinaryTree(newNode)
            self.left = tree
            tree.left = self.left


def printTree(tree):
        if tree != None:
            printTree(tree.getLeftChild())
            print(tree.getNodeValue())
            printTree(tree.getRightChild())

# test tree
def testTree():
    myTree = BinaryTree("Maud")
    myTree.insertLeft("Bob")
    myTree.insertRight("Tony")
    myTree.insertRight("Steven")
    printTree(myTree)
testTree()
```

tly more difficult level utilizing multiple concepts that you’ve learned in the previous sections.

47. Design a basic game in which a robot starting from point (0,0), moves as you tell it to. The available commands will be UP, DOWN, RIGHT and LEFT. Using the formula for distance between two points, calculate the distance from the origin to the position of the robot after giving your command.
Input format:
UP 6
DOWN 2
LEFT 2
RIGHT 7
#calculates distance from (0,0) to (5,4)
#answer should be 6

```
import math
pos = [0,0]
for i in range(4):
    s = input()
    if not s:
        break
    movement = s.split(" ")
    direction = movement[0]
    steps = int(movement[1])
    if direction=="UP":
        pos[0]+=steps
    elif direction=="DOWN":
        pos[0]-=steps
    elif direction=="LEFT":
        pos[1]-=steps
    elif direction=="RIGHT":
        pos[1]+=steps
    else:
        pass

print(int(round(math.sqrt(pos[1]**2+pos[0]**2))))
```

48. The next program will be an introduction to GUI programming to give you an idea how you can design front end of your program to be more visually appealing. We will use the tkinter module to design a simple calculator.

```
"""
calculator has a layout like this ...
<   display   >
7  8  9  *  C
4  5  6  /  M->
1  2  3  -  ->M
0  .  =  +  neg
"""
import tkinter as tk
def click(key):
    global memory
    if key == '=':
        # avoid division by integer
        if '/' in entry.get() and '.' not in entry.get():
            entry.insert(tk.END, ".0")
        # guard against the bad guys abusing eval()
        str1 = "-+0123456789."
        if entry.get()[0] not in str1:
            entry.insert(tk.END, "first char not in " + str1)
        # here comes the calculation part
        try:
            result = eval(entry.get())
            entry.insert(tk.END, " = " + str(result))
        except:
            entry.insert(tk.END, "--> Error!")
```

```
    elif key == 'C':
        entry.delete(0, tk.END)  # clear entry
    elif key == '->M':
        memory = entry.get()
        # extract the result
        if '=' in memory:
            ix = memory.find('=')
            memory = memory[ix+2:]
        root.title('M=' + memory)
    elif key == 'M->':
        entry.insert(tk.END, memory)
    elif key == 'neg':
        if '=' in entry.get():
            entry.delete(0, tk.END)
        try:
            if entry.get()[0] == '-':
                entry.delete(0)
            else:
                entry.insert(0, '-')
        except IndexError:
            pass
    else:
        # previous calculation has been done, clear entry
        if '=' in entry.get():
            entry.delete(0, tk.END)
```

```
        entry.insert(tk.END, key)
root = tk.Tk()
root.title("Simple Calculator")
btn_list = [
'7',  '8',  '9',  '*',  'C',
'4',  '5',  '6',  '/',  'M->',
'1',  '2',  '3',  '-',  '->M',
'0',  '.',  '=',  '+',  'neg' ]
# create all buttons with a loop
r = 1
c = 0
for b in btn_list:
    rel = 'ridge'
    cmd = lambda x=b: click(x)
    tk.Button(root,text=b,width=5,relief=rel,command=cmd).grid(row=r,column=c)
    c += 1
    if c > 4:
        c = 0
        r += 1
# use Entry widget for an editable display
entry = tk.Entry(root, width=33, bg="yellow")
entry.grid(row=0, column=0, columnspan=5)
root.mainloop()
```

49. Create a simple number guessing game.

```
    else: break                     # yay!

print("That took",guesses,"guesses")
binary=False                    # set this to True or False
lonum,hinum=1,128               # range for the number

import random as r

the_num=r.randint(lonum,hinum)  # computer chooses a number randomly
print("I'm thinking of a number between",lonum,"and",hinum)

lo=1
hi=hinum
guesses=0

for i in range(lonum,hinum):    # repeat this until guess is correct:

    guess=int(input ("What is your guess: "))
    if binary:  guess=lo+(hi-lo)//2
    else:       guess=r.randint(lo,hi)
    print("Guess:",guess)
    guesses+=1                      # add 1 to count of guesses
                                    # check the guessed number
    if guess > the_num:
        print("Lower!")
        hi=guess                    # bring down the upper bound
    elif guess < the_num:
        print("Higher!")
        lo=guess                    # push up the lower bound
```

50. For the final program, we're going to design a complete tic-tac-toe game. You will play against the computer. The program has been explained with comments and the function nomenclature makes it intuitive to understand. Many functions you learned has been incorporated into this program.

```
# Tic Tac Toe
import random
def drawBoard(board):
    # This function prints out the board that it was passed.
    # "board" is a list of 10 strings representing the board (ignore index 0)
    print('   |   |')
    print(' ' + board[7] + ' | ' + board[8] + ' | ' + board[9])
    print('   |   |')
    print('-----------')
    print('   |   |')
    print(' ' + board[4] + ' | ' + board[5] + ' | ' + board[6])
    print('   |   |')
    print('-----------')
    print('   |   |')
    print(' ' + board[1] + ' | ' + board[2] + ' | ' + board[3])
    print('   |   |')

def inputPlayerLetter():
    # Lets the player type which letter they want to be.
    # Returns a list with the player's letter as the first item, and the computer's letter as the second.
    letter = ''
    while not (letter == 'X' or letter == 'O'):
        print('Do you want to be X or O?')
        letter = input().upper()
        # the first element in the list is the player's letter, the second is the computer's letter.
    if letter == 'X':
        return ['X', 'O']
    else:
        return ['O', 'X']
```

```
30
31  def whoGoesFirst():
32      # Randomly choose the player who goes first.
33      if random.randint(0, 1) == 0:
34          return 'computer'
35      else:
36          return 'player'
37
38  def playAgain():
39      # This function returns True if the player wants to play again, otherwise it returns False.
40      print('Do you want to play again? (yes or no)')
41      return input().lower().startswith('y')
42
43  def makeMove(board, letter, move):
44       board[move] = letter
45
46  def isWinner(bo, le):
47       # Given a board and a player's letter, this function returns True if that player has won.
48       # We use bo instead of board and le instead of letter so we don't have to type as much.
49       return ((bo[7] == le and bo[8] == le and bo[9] == le) or # across the top
50       (bo[4] == le and bo[5] == le and bo[6] == le) or # across the middle
51       (bo[1] == le and bo[2] == le and bo[3] == le) or # across the bottom
52       (bo[7] == le and bo[4] == le and bo[1] == le) or # down the left side
53       (bo[8] == le and bo[5] == le and bo[2] == le) or # down the middle
54       (bo[9] == le and bo[6] == le and bo[3] == le) or # down the right side
55       (bo[7] == le and bo[5] == le and bo[3] == le) or # diagonal
56       (bo[9] == le and bo[5] == le and bo[1] == le)) # diagonal
57
```

```
def getBoardCopy(board):
     # Make a duplicate of the board list and return it the duplicate.
    dupeBoard = []
    for i in board:
        dupeBoard.append(i)
    return dupeBoard

def isSpaceFree(board, move):
    # Return true if the passed move is free on the passed board.
    return board[move] == ' '

def getPlayerMove(board):
    # Let the player type in their move.
    move = ' '
    while move not in '1 2 3 4 5 6 7 8 9'.split() or not isSpaceFree(board, int(move)):
        print('What is your next move? (1-9)')
        move = input()
    return int(move)

def chooseRandomMoveFromList(board, movesList):
    # Returns a valid move from the passed list on the passed board.
    # Returns None if there is no valid move.
    possibleMoves = []
    for i in movesList:
        if isSpaceFree(board, i):
            possibleMoves.append(i)
    if len(possibleMoves) != 0:
        return random.choice(possibleMoves)
    else:
```

```
        return None

def getComputerMove(board, computerLetter):
    # Given a board and the computer's letter, determine where to move and return that move.
    if computerLetter == 'X':
        playerLetter = 'O'
    else:
        playerLetter = 'X'

    # Here is our algorithm for our Tic Tac Toe AI:
    # First, check if we can win in the next move
    for i in range(1, 10):
        copy = getBoardCopy(board)
        if isSpaceFree(copy, i):
            makeMove(copy, computerLetter, i)
            if isWinner(copy, computerLetter):
                return i

    # Check if the player could win on their next move, and block them.
    for i in range(1, 10):
        copy = getBoardCopy(board)
        if isSpaceFree(copy, i):
            makeMove(copy, playerLetter, i)
            if isWinner(copy, playerLetter):
                return i

    # Try to take one of the corners, if they are free.
    move = chooseRandomMoveFromList(board, [1, 3, 7, 9])
    if move != None:
```

```
116             return move
117
118         # Try to take the center, if it is free.
119         if isSpaceFree(board, 5):
120             return 5
121
122         # Move on one of the sides.
123         return chooseRandomMoveFromList(board, [2, 4, 6, 8])
124
125     def isBoardFull(board):
126         # Return True if every space on the board has been taken. Otherwise return False.
127         for i in range(1, 10):
128             if isSpaceFree(board, i):
129                 return False
130         return True
131
132
133     print('Welcome to Tic Tac Toe!')
134
135     while True:
136         # Reset the board
137         theBoard = [' '] * 10
138         playerLetter, computerLetter = inputPlayerLetter()
139         turn = whoGoesFirst()
140         print('The ' + turn + ' will go first.')
141         gameIsPlaying = True
142
143         while gameIsPlaying:
144             if turn == 'player':
```

```
            # Player's turn.
            drawBoard(theBoard)
            move = getPlayerMove(theBoard)
            makeMove(theBoard, playerLetter, move)

            if isWinner(theBoard, playerLetter):
                drawBoard(theBoard)
                print('Hooray! You have won the game!')
                gameIsPlaying = False
            else:
                if isBoardFull(theBoard):
                    drawBoard(theBoard)
                    print('The game is a tie!')
                    break
                else:
                    turn = 'computer'

        else:
            # Computer's turn.
            move = getComputerMove(theBoard, computerLetter)
            makeMove(theBoard, computerLetter, move)

            if isWinner(theBoard, computerLetter):
                drawBoard(theBoard)
                print('The computer has beaten you! You lose.')
                gameIsPlaying = False
            else:
                if isBoardFull(theBoard):
                    drawBoard(theBoard)
                    print('The game is a tie!')
                    break
                else:
                    turn = 'player'

    if not playAgain():
        break
```

Python tips and tricks:

51. When to take your programming to the next level with complex structures and algorithms, maintainability and readability is often lost. This is frustrating when working on a repository such a GitHub with other programmers who do not follow conventions. Fortunately, the official Python has released a documentation of good coding practices and conventions to follow. Go through the following link: **https://www.python.org/dev/peps/pep-0008/**

52. In order to swap the values of variables, unlike other languages Python can do this in a single line (Ex 1). This also works for consecutive pairs of elements in any mutable set of values (Ex 2).

```
x, y = y, x

list[i], list[i + 1] = list[i + 1], list[i]
```

53. Initializing a list with a value multiple times is very simple.

```
items=[0]*3
print(items)

>>>[0,0,0]
```

54. Converting a list to a string.

```
fruits = ["Apple", "Banana", "Kiwi", "Watermelon"]
print(", ".join(fruits))

>>>'Apple, Banana, Kiwi, Watermelon'
```

55. Sometimes you will need to operate on only a portion of the list. Here are a few ways to do that.

Note: List ranges always include initial value and stop one increment short of the final value in the range. List ranges also begin with 0 and count upwards.

```
x = [1,2,3,4,5,6]
#First 3
#Printing all values from the beginning of the list range upto but not including the 5th value
print(x[:3])
>>> [1,2,3]

#Middle 4
#Printing all values starting from the second value going up to the sixth value
print(x[1:4])
>>> [2,3,4,5]

#Last 3
print(x[-3:])
>>> [4,5,6]

#Odd numbers
print(x[::2])
>>> [1,3,5]

#Even numbers
print(x[1::2])
>>> [2,4,6]
```

56. Python is known for very short and intuitive code. Here is a simple inline if statement which will take you far if mastered.

```
print("Hello" if True else "World")
>>>Hello
```

57. In addition to python's built in datatypes they also include a few extra for special use cases in the collections module. The Counter is quite useful on occasion.

```
from collections import Counter
print(Counter("hello"))
>>>Counter({'l': 2, 'h': 1, 'e': 1, 'o': 1})
```

58. Along with the collections library python also has a library called itertools which has really cool efficient solutions to problems. One is finding all combinations. This will tell us all the different ways the teams can play each other.

```
from itertools import combinations

teams = ["Packers", "49ers", "Ravens", "Patriots"]
for game in combinations(teams, 2):
    print(game)

>>> ('Packers', '49ers')
>>> ('Packers', 'Ravens')
>>> ('Packers', 'Patriots')
>>> ('49ers', 'Ravens')
>>> ('49ers', 'Patriots')
>>> ('Ravens', 'Patriots')
```

59. To reverse a string/list/tuple you can try the following:

```
word = "martin"
print("reverse is " + word[::-1])
```

60. Extended unpacking a list in python3.

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

>>>[1]
>>>[2,3,4]
>>>[5]
```

61. In the Python console, whenever we test an expression or call a function, the result dispatches to a temporary name, _ (an underscore).

```
>>> 2 + 1
3
>>> _
3
>>> print _
3
```

62. Like we use list comprehensions, we can also use dictionary/set comprehensions. They are simple to use and just as effective.

```
testDict = {i: i * i for i in range(10)}
testSet = {i * 2 for i in range(10)}

print(testSet)
print(testDict)

#set([0, 2, 4, 6, 8, 10, 12, 14, 16, 18])
#{0: 0, 1: 1, 2: 4, 3: 9, 4: 16, 5: 25, 6: 36, 7: 49, 8: 64, 9: 81}
```

63. To verify multiple values, we can do in the following manner.

```
if m in [1,3,5,7]:

instead of:
if m==1 or m==3 or m==5 or m==7:
```

64. Four Ways To Reverse String/List.

```
# Reverse The List Itself.
testList = [1, 3, 5]
testList.reverse()
print(testList)
>>> [5, 3, 1]

# Reverse While Iterating In A Loop.
for element in reversed([1,3,5]): print(element)
>>> 5
>>> 3
>>> 1

# Reverse A String In Line.
"Test Python"[::-1]
>>>"nohtyP tseT"

# Reverse A List Using Slicing.
[1, 3, 5][::-1]
>>>[5,3,1]
```

65. Python3 doesn’t contain a method to create a switch-case statement unlike c/c++. This is an alternative.

```
def xswitch(x):
     return xswitch._system_dict.get(x, None)

xswitch._system_dict = {'files': 10, 'folders': 5, 'devices': 2}

print(xswitch('default'))
print(xswitch('devices'))

>>> None
>>> 2
```

66. When working with iterables, you can use the following technique to shorten your code.

```
#Instead of doing:

i = 0
for item in iterable:
    print i, item
    i += 1

#We can do:
for i, item in enumerate(iterable):
    print i, item
```

67. To check if two words are anagrams, use the counter function.

```
from collections import Counter
def is_anagram(str1, str2):
     return Counter(str1) == Counter(str2)
is_anagram('abcd','dbca')
>>>True
is_anagram('abcd','dbaa')
>>>False
```

68. Instead of using the join function to make a string out of list elements, you can try the following method.

```
row = ["24", "jon", "sdk", "python"]
#instead of this:
print(','.join(str(x) for x in row))
>>> 24,jon,sdk,python
#try this:
print(*row, sep=',')
>>> 24,jon,sdk,python
```

69. You can use functions like sum to make greater generators, reducing the number of lines of code.

```
#instead of this:
sum = 0
for i in range(1300):
    if i % 3 == 0 or i % 5 == 0:
        sum += i
print(sum)
#try this:
sum(i for i in range(1300) if i % 3 == 0 or i % 5 == 0)
```

70. Perhaps the easiest way of creating a tree is with one simple line.

```
from collections import defaultdict
def tree(): return defaultdict(tree)
#says that a tree is a dict whose default values are trees.
users = tree()
users['harold']['username'] = 'hrldcpr'
users['handler']['username'] = 'matthandlersux'

>>>{"harold": {"username": "hrldcpr"}, "handler": {"username": "matthandlersux"}}
```

71. Creating a unified list without loops.

```
import itertools
test = [[-1, -2], [30, 40], [25, 35]]
print(list(itertools.chain.from_iterable(test)))

>>> [-1, -2, 30, 40, 25, 35]
```

72. Check the memory usage of an object by doing this.

```
import sys
x=1
print(sys.getsizeof(x))

>>> 28
```

73. You can declare multiple variables to call the same function in a line.

```
# function returning multiple values.
def x():
    return 1, 2, 3, 4
# Calling the above function.
a, b, c, d = x()
print(a, b, c, d)
>>> 1 2 3 4
```

74. Unpack Function Arguments Using Splat Operator. (*)

```
def test(x, y, z):
    print(x, y, z)

testDict = {'x': 1, 'y': 2, 'z': 3}
testList = [10, 20, 30]

test(*testDict)
test(**testDict)
test(*testList)

>>> x y z
>>> 1 2 3
>>> 10 20 30
```

75. Using the reduce function to calculate the factorial of any number in a single line.

```
import functools
result = (lambda k: functools.reduce(int.__mul__, range(1,k+1),1))(3)
print(result)
>>>6
```

76. Complex programs require a handy built in debugging tool. In this case, We can set breakpoints in our Python script with the help of the ‘pdb’ module.

```
import pdb
pdb.set_trace()
```

77. You can inspect an object and see all the operations available to perform on that object by using the dir() method.

```
test = [1, 3, 5, 7]
print( dir(test) )
>>>['__add__', '__class__', '__contains__', '__delattr__', '__delitem__',
'__delslice__', '__doc__', '__eq__', '__format__', '__ge__', '__getattribute__',
'__getitem__', '__getslice__', '__gt__', '__hash__', '__iadd__', '__imul__',
'__init__', '__iter__', '__le__', '__len__', '__lt__', '__mul__', '__ne__',
'__new__', '__reduce__', '__reduce_ex__', '__repr__', '__reversed__', '__rmul__',
'__setattr__', '__setitem__', '__setslice__', '__sizeof__', '__str__',
'__subclasshook__', 'append', 'count', 'extend', 'index', 'insert', 'pop', 'remove',
'reverse', 'sort']
```

78. Use a dictionary to store expressions and perhaps use as a primitive switch statement.

```
stdcalc = {
    'sum': lambda x, y: x + y,
    'subtract': lambda x, y: x - y
}

print(stdcalc['sum'](9,3))
print(stdcalc['subtract'](9,3))

>>> 12
>>> 6
```

79. If you have used the eval() function, you should know it has its dangers and there is a safer option. It evaluates the code as soon as the function is called. ‘ast.literal_eval’ raises an exception if the input isn't a valid Python datatype, so the code won't be executed if it's not.

```
import ast
datamap = ast.literal_eval(input('Provide some data here: '))
```

80. This last tip will be a list of the most helpful Youtube channels: ‘sentdex’,’thenewboston’,’Python training by Dan Bader’,’Corey Schafer’,’Clever Programmer’,’Trevor Payne’.

Python3 modules:

The following modules are of two types: those offered along with the standard package and those offered by third party organizations that make these libraries open source. These modules have been selected based on their usefulness and popularity. They are categorized based on which major industry uses them. You will find that a variety of purposes can be fulfilled with these modules. Python is a general purpose programming language and we would like to show you few of the modules that aid different purposes for all your needs.

1. The Data Science modules:

The three core modules used for scientific computing are numpy, scipy and matplotlib. They offer a variety of tools for graphing to trigonometric function to computing differential equations. If you wish to become a data scientist this is where you should start. Extended learning: Pandas and IPython provides additional tools for data science.

(i)NumPy module:

NumPy introduces objects for multidimensional arrays and matrices, as well as routines that allow developers to perform advanced mathematical and statistical functions on those arrays with as little code as possible.

(ii)SciPy module:

It builds on NumPy by adding a collection of algorithms and high-level commands for manipulating and visualizing data. This package includes functions for computing integrals numerically, solving differential equations, optimization, and more.

You can find all the functions provided in the official documentation provided below.

link : https://docs.scipy.org/doc/numpy/reference/routines.math.html

(iii)MatPlotLib module:

Used for creating 2D plots and graphs. It’s relatively low-level, meaning it requires more commands to generate nice-looking graphs and figures than with some more advanced libraries. However, with enough commands, you can make just about any kind of graph you want with matplotlib**.**

2. The Machine Learning modules:

By training a computer to read and interpret real world data, we can create algorithms that make more accurate predictions. Machine Learning sits on the border of Artificial Intelligence and Statistical Analysis. These modules offer common algorithms to work with ML such as regression algorithms and methods for neural networks.

(i) scikit-learn:
It builds on NumPy and SciPy by adding a set of algorithms for common machine learning and data mining tasks, including clustering, regression, and classification.
(ii) Theano:
It uses NumPy-like syntax to optimize and evaluate mathematical expressions. What sets Theano apart is that it takes advantage of the computer's GPU in order to make data-intensive calculations up to 100x faster than the CPU alone. Theano's speed makes it especially valuable for deep learning and other computationally complex tasks.
(iii) TensorFlow:
It is another high-profile entrant into machine learning, developed by Google as an open-source successor to DistBelief, their previous framework for training neural networks. TensorFlow uses a system of multi-layered nodes that allow you to quickly set up, train, and deploy artificial neural networks with large datasets. It's what allows Google to identify objects in photos or understand spoken words in its voice-recognition app.

3. The re module(regex):
This is the regular expressions module. It offers all the same syntax as perl, UNIX and other languages. A regular expression is a special sequence of characters that helps you match or find other strings or sets of strings, using a specialized syntax.
Important functions:
(i) match function: re.match(pattern, string, flags=0)

(ii) search function: re.search(pattern, string, flags=0)
(iii) sub function: re.sub(pattern, repl, string, max=0)

4. The sys module:

The sys module allows you to use stdin() and stdout(), as well as stderr(), but, most interestingly, we can utilize sys.argv(). The idea of sys.argv is to allow you to pass script arguments through to Python from the command line.

5. The os module:

The OS module in Python provides a way of using operating system dependent functionality. The functions that the OS module provides allows you to interface with the underlying operating system that Python is running on (Windows, Mac or Linux). You can find important information about your location or about the process.
Important functions:
Executing a shell command:
os.system()
Get the users environment:
os.environ()
Return information identifying the current operating system:
os.uname()
Change the root directory of the current process to path:
os.chroot(path)
Return a list of the entries in the directory given by path:
os.listdir(path)
Create a directory named path with numeric mode mode:
os.mkdir(path)

6. The collections module:

This module implements unique container datatypes providing alternatives to Python’s built-in containers dict, list, set, and tuple.
Examples: defaultdict, OrderedDict, counter, deque, namedtuple, enum.Enum

7. The itertools module:

Itertools is a module for building iterators. It is part of the Python Standard Library. The tools provided by itertools are fast and memory efficient. You will be able to take these building blocks to create your own specialized iterators that can be used for efficient looping.
Important functions:
count(no.), islice(iterable, stop), ifilter(predicate, iterable),
imap(function, *iterables)

8. The urllib module:

The urllib module in Python 3 allows you access websites via your program. Through urllib, you can access websites, download data, parse data and modify your headers. Some websites do not appreciate programs accessing their data and placing weight on their servers. When they find out that a program is visiting them, they may sometimes choose to block you out, or serve you different data that a regular user might see. This can be annoying at first, but can be overcome with some simple code.

9. The threading module:

It’s part of the standard library. This module is used to run multiple processes on python at the same time as opposed to its linear nature.

10. The tkinter module:

This is part of the standard library. It contains a toolkit to create cross-platform GUI. You can create widgets, windows, checkboxes, radio buttons, text boxes, etc. You can connect it to databases and modify user entered information. The tkinter module is perhaps the most widely used basic GUI library although you can take a look at wxpython, pyGtk and pyQT modules as well.

11. The pygame module:

This library will help you in 2d game development. It contains a variety of tools to create simple games such as chess, the snake game, tic-tac-toe, etc.

12. The pyglet module:

A powerful 3d animation and game creation engine. This is the engine in which the famous minecraft was made.

13. The sh module:

Not available in standard library and needs to be installed.
sh is a unique subprocess wrapper that maps your system programs to Python functions dynamically. sh helps you write shell scripts in Python by giving you the good features of Bash (easy command calling, easy piping) with all the power
and flexibility of Python.

14. The pymysql module:

This is a third party module and not included in the standard library. It is used for ORM (object relational mapping) and provides a database connection to mysql. Alternatives are provided by SQLAlchemy. This is essential to python software development.

15. The pycrypto module:

This is the cryptography module. If you are looking for a job in cybersecurity, this is a must-learn. It provides methods of creating AES 128 bit encryption, key generators, ciphers, etc.

16. The socket module:

It basically provides access to network communication. The socket module exposes the low-level C API for communicating over a network using the BSD socket interface. It includes the socket class, for handling the actual data channel, and functions for network-related tasks such as converting a server’s name to an address and formatting data to be sent across the network.

17. The BeautifulSoup module:

Beautiful Soup is a Python library for pulling data out of HTML and XML files. The urllib module is often used in combination with this module. It is used for web scraping and parsing documents. You can also use the Scrapy module for web scraping.

18. The Twisted module:

The most important tool for any network application developer. It has a very beautiful api and is used by a lot of famous python developers.

19. The Pillow module:

PIL is the python cross-platform imaging library and a must have for anyone who works with images. Some of the file formats supported include PPM, PNG, JPEG, GIF, TIFF, and BMP. It is also possible to create new file decoders to expand the library of file formats accessible.

20. The nose framework:

If you’re into testing in python, this is the framework to use. Do check it out. Used for automated testing and test driven development.

Closing note:
This book was written keeping in mind the needs of a beginner; to truly make the journey much easier. Please experiment with the provided code, tips, tricks and examples to become a seasoned Pythonista yourself. Learning Python has no end. Enjoy the journey and don’t be anxious to reach your destination. Good luck!