introduction

* python coding can be done in 3 different ways
  + immediate mode
  + script mode
  + IDE (Vscode, Jupyterlab, Jupyter notebook, pycharms, etc)

constants

* we use uppercase when naming constants
* Ex: PI = 3.14

Comments

* Use # to add comments (Hash symbol, octothorpe, number sign)
* Python doesn’t have multiline comments

Docstrings

* Documentation strings
* Designed to explain code
* Use “”” “””
* Used below the function’s 1st line to explain the function’s use
* def shout(word):

    """

    Print a word with an

    exclamation mark following it.

    """

    print(word + "!")

shout("spam")

* Programmer can inspect docstrings at runtime - unlike comments

Float

* ‘/’ returns a float value

|  |  |
| --- | --- |
| print(8/2) | 4.0 |

Exponentiation

* If the power is a partition it will output a float

|  |  |
| --- | --- |
| print(2\*\*3) | 8 |
| print(2\*\*2\*\*3) | 256 |
| print(2\*\*3\*\*2) | 512 |
| print((2\*\*2)\*\*3) | 64 |
| print(9\*\*(1/2)) | 3.0 |

Quotient (Integer division)

|  |  |
| --- | --- |
| print(20//6) | 3 |
| print(20.0//6) | 3.0 |
| print(1000//1.6) | 624.0 |
| print(10//4) | 2 |

Remainder

|  |  |
| --- | --- |
| print(20%6) | 2 |
| print(1.25%0.5) | 0.25 |
| print(7%(5//2)) | 1 |

Strings

* Strings can be indexed like lists
* Use backlash “\” to skip when printing characters such as ’

|  |  |
| --- | --- |
| print(“I\’m Dilshan”) | I’m Dilshan |

Newlines

|  |  |
| --- | --- |
| print("Hello\nworld") | Hello  world |
| print("Hello\tworld") | Hello world |
| print("""This  is  a  multiline  text""") | This  is  a  multiline  text |
| print('A \nB \nC \nD') | A  B  C  D |

String operations

* Concatenation using ‘+’
* print("A"+'B')
* o/p: AB
* HackerRank: Mutations

**def** mutate\_string(string, position, character):

    string = string[:position] + character + string[position + 1:]

**return** string

**if** \_\_name\_\_ == '\_\_main\_\_':

    s = **input**()

    i, c = **input**().split()

    s\_new = mutate\_string(s, **int**(i), c)

**print**(s\_new)

* Using , will add a space
* print(“A”,”B”)
* o/p: A B
* String multiplication using ‘\*’
  + print("Hi"\*3)
  + Output: HiHiHi
* Ex:
  + print(3\*"Hi")
  + Output: HiHiHi
* Ex:
  + print(4\*'2')
  + Output: 2222
* Strings can’t be multiplied by other strings
* Strings also can’t be multiplied by floats, even if the integers are whole numbers
* Use str(x) to convert number to String - Can be used for concatenation
* txt = “cat”

print(txt.center(10, ”-”))

* Above code centers text “cat” while filling left and right with “-” as full length is 10
* count(str): Number of times str appearing in text
* text = input()

letter = input()

counter = text.count(letter)

print(int((counter\*100)/len(text)))

* upper() and lower()
* HackerRank: Find a string

**def** count\_substring(string, sub\_string):

    count = 0

**for** i **in** **range**(**len**(string)):

**if** string[i : **len**(sub\_string) + i] == sub\_string:

            count += 1

**return** count

**if** \_\_name\_\_ == '\_\_main\_\_':

    string = **input**().strip()

    sub\_string = **input**().strip()

    count = count\_substring(string, sub\_string)

**print**(count)

String validators

* str.isalnum() -> checks if all characters are alphanumeric
* str.isalpha() -> checks if all characters are alphabetical
* str.isdigit() -> checks if all characters are digits
* str.islower() -> checks if all characters are lowercase
* str.isupper() -> checks if all characters are uppercase
* HackerRank: String validators

**if** \_\_name\_\_ == '\_\_main\_\_':

    s = **input**()

    alnum = 0

    alpha = 0

    digit = 0

    lower = 0

    upper = 0

**for** i **in** s:

**if** i.isalnum() == **True**:

            alnum += 1

**if** i.isalpha() == **True**:

            alpha += 1

**if** i.isdigit() == **True**:

            digit += 1

**if** i.islower() == **True**:

            lower += 1

**if** i.isupper() == **True**:

            upper += 1

**print**(**bool**(alnum))

**print**(**bool**(alpha))

**print**(**bool**(digit))

**print**(**bool**(lower))

**print**(**bool**(upper))

Text alignment

* String of text can be aligned left, right or center
* .ljust(width) -> Left align
* .center(width) -> centered
* .rjust(width) -> right align
* We can fill it with any other string
* Ex: “car”.ljust(3, \*)

o/p: car\*\*\*

* HackerRank: Text Wrap

**import** textwrap

**def** wrap(string, max\_width):

**for** i **in** **range**(0, **len**(string), max\_width):

**print**(string[i:max\_width + i])

**return** ""

**if** \_\_name\_\_ == '\_\_main\_\_':

    string, max\_width = **input**(), **int**(**input**())

    result = wrap(string, max\_width)

**print**(result)

* HackerRank: String formatting

**def** print\_formatted(number):

    l = **len**(**bin**(number)[2:])

**for** i **in** **range** (1, number +1 ):

**print**(**str**(i).rjust(l),**str**(**oct**(i))[2:].rjust(l),**str**(**hex**(i))[2:].upper().rjust(l),**str**(**bin**(i)[2:]).rjust(l))

**if** \_\_name\_\_ == '\_\_main\_\_':

    n = **int**(**input**())

    print\_formatted(n)

replace

* text = “CAT”

text = text.replace(“C”,”B”)

print(text)

* o/p: BAT
* Below code is for find/replace app
* text = "Amy has 128 dollars, while David has 54 dollars. Amy is going to the zoo. David watches soccer."

find = input()

replace = input()

print(text.count(find))

print(text.replace(find, replace))

Variables

* Allowed variable characters - Letters, numbers and underscores
* Names can’t start with numbers
* Python is case sensitive
* Python variables don’t have specific data types

Taking user input

* Use ‘input()’
* The user input is taken as a string
* Use int(input()) to convert user input to int
* str(42) gives string 42
* use float() to convert to float

In place and walrus operators

* x = x+3 is same as x+=3
* Also can be used on strings
* Same thing can be done with these (+, -, \*, /, %, \*\*, //)
* Walrus operator :=
  + print(num:=int(input()))
  + This is same as,

num = int(input())

print(num)

Boolean

* True and False
* Use == and != to compare
* Comparison operators are called relational operators
* > < can also be used to compare different types such as int and float
* lexicographically - the alphabetical order of words is based on the alphabetical order of their component letters
  + print("Annie" > "Andy")
  + Output - True
  + Compared letter by letter (n>d)

If statements

* Ex:
  + temp = int(input())

If (temp >= 100):

print(“Boiling”)

Boolean

* ‘and’ gets priority over ‘or’

List

* Lists can contain data from different data types
* Lists can be nested
* HackerRank: NestedLists

if \_\_name\_\_ == '\_\_main\_\_':

students = []

scores = []

for \_ in range(int(input())):

name = input()

score = float(input())

students.append([name,score])

scores.append(score)

scores = set(scores)

students.sort()

scores.remove(min(scores))

secMin = min(scores)

for i in range(0, len(students)):

mark = students[i][1]

if secMin == mark:

print(students[i][0])

* Lists can be added and multiplied

|  |  |
| --- | --- |
| nums = [1, 2, 3]  print(nums + [4, 5, 6])  print(nums \* 3) | [1, 2, 3, 4, 5, 6]  [1, 2, 3, 1, 2, 3, 1, 2, 3] |

* Use ‘in’ operator to check if an item is in the list
* words = ["spam", "egg", "spam", "sausage"]

print("spam" in words) → True

* ‘in’ can also be used to check if a string is a substring of another string

items = [42, 88, 721, 12, 43, 22, 908]

num = int(input())

if (num in items):

    print("bingo") → bingo

* Use ‘not’ operator to check if an item isn’t in the list
* nums = [1, 2, 3]

print(not 4 in nums) → True

List functions

* append adds items to the end of list
* nums = [1, 2, 3]

nums.append(4)

print(nums)

* o/p: [1, 2, 3, 4]
* append is a method in list class
* Use len to get the number of items in a list
* nums = [1, 3, 5, 2, 4]

print(len(nums))

* o/p: 5
* Below code gives middle item’s index
* items = [2, 4, 6, 8, 10, 12, 14]

print(len(items)//2)

* insert allows to replace index
* Items after inserted item shifts to right
* words = ["Python", "fun"]

index = 1

words.insert(index, "is")

print(words)

* words = ["Python", "fun"]

words.insert(1, "is")

print(words)

* o/p: ['Python', 'is', 'fun']
* index finds the first occurrence of item and returns index
* if item is not in list returns ValueError
* letters = ['p', 'q', 'r', 's', 'p', 'u']

print(letters.index('r'))

print(letters.index('p'))

print(letters.index('z'))

* o/p:

2

0

Traceback (most recent call last):

File "file0.py", line 4, in <module>

print(letters.index('z'))

ValueError: 'z' is not in list

* max(list) is used to find maximum value
* min(list) is used to find minimum value
* list.count(item) counts the number of occurances of item
* list.remove(item) removes object from list

Reverse

* list.reverse() reverse items in a list
* HackerRank: Array - DS

*#!/bin/python3*

**import** math

**import** os

**import** random

**import** re

**import** sys

**def** reverseArray(a):

**return** **reversed**(a)

**if** \_\_name\_\_ == '\_\_main\_\_':

    fptr = **open**(os.environ['OUTPUT\_PATH'], 'w')

    arr\_count = **int**(**input**().strip())

    arr = **list**(**map**(**int**, **input**().rstrip().split()))

    res = reverseArray(arr)

    fptr.write(' '.join(**map**(**str**, res)))

    fptr.write('\n')

    fptr.close()

More list functions

* all and any - Take a list as argument and return true if all or any
* enumerate - Used to iterate through values and indices of a list simultaneously
* nums = [55, 44, 33, 22, 11]

if all([i > 5 for i in nums]):

    print("All larger than 5")

if any([i % 2 == 0 for i in nums]):

    print("At least one is even")

for v in enumerate(nums):

    print(v)

* o/p:

All larger than 5

At least one is even

(0, 55)

(1, 44)

(2, 33)

(3, 22)

(4, 11)

* pop(index) remove item at given index
* remove(item) remove given item
* count(item) returns number of times item occurs in list
* list.reverse()
* list.sort()
* max(list) and min(list)
* prices = [125000, 78000, 110000, 65000, 300000, 250000, 210000, 150000, 165000, 140000, 125000, 85000, 90000, 128000, 230000, 225000, 100000, 300000]

avg = sum(prices)/len(prices)

count = 0

for i in prices:

    if i > avg:

        count += 1

print(count)

* HackerRank: Lists

if \_\_name\_\_ == '\_\_main\_\_':

    N = int(input())

    list\_ = []

    n = 0

    command = []

    while n < N:

        input\_ = input()

        if input\_[:6] == "insert":

            command = input\_.split()

            list\_.insert(int(command[1]), int(command[2]))

        elif input\_[:5] == "print":

            print(list\_)

        elif input\_[:6] == "remove":

            command = input\_.split()

            list\_.remove(int(command[1]))

        elif input\_[:6] == "append":

            command = input\_.split()

            list\_.append(int(command[1]))

        elif input\_[:4] == "sort":

            list\_.sort()

        elif input\_[:3] == "pop":

            command = input\_.split()

            list\_.pop(-1)

        else:

            list\_.reverse()

        n += 1

While

* i = 1

while i <=5:

    print(i)

    i = i + 1

print("Finished!")

* items = int(input())

days = int(input())

count = items

i = 0

while (i < days):

    count \*= 2

    i += 1

print(count)

* x = 1

while x < 10:

    if x%2 == 0:

        print(str(x) + " is even")

    else:

        print(str(x) + " is odd")

    x += 1

* Use break to break the loop
* i = 0

while 1==1:

    print(i)

    i = i + 1

    if i >= 5:

        print("Breaking")

        break

print("Finished")

* while True makes an infinite loop
* break can’t be used outside loop
* Use continue to skip a loop and jump back to the beginning
* i = 0

while i<5:

  i += 1

  if i==3:

    print("Skipping 3")

    continue

  print(i)

* continue also can’t be used outside loop
* Used when number of iterations are unknown

for loop

* words = ["hello", "world", "spam", "eggs"]

for word in words:

    print(word + "!")

* Can use to iterate over strings
* str = "testing for loops"

count = 0

for x in str:

   if(x == 't'):

    count += 1

print(count)

* break and continue can be used inside for loop
* Used when number of iterations are known
* list = [1, 2, 3, 4, 5, 6, 7, 8, 9]

sum = 0

for x in list:

    sum += x

print(sum)

range()

* Returns a sequence of numbers
* Default - starts with 0, increases by 1
* Use list() to convert it to a list
* numbers = list(range(10))

print(numbers)

* o/p: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
* numbers = list(range(3, 8))

print(numbers)

print(range(20) == range(0, 20))

* o/p:

[3, 4, 5, 6, 7]

True

* third argument in range define the interval (step)
* numbers = list(range(5, 20, 2))

print(numbers)

* o/p: [5, 7, 9, 11, 13, 15, 17, 19]

for loop with range

* for i in range(5):

    print("hello!")

* Here, range(5) isn’t a list - Just a sequence
* a = int(input())

b = int(input())

listA = list(range(a,b))

print(listA)

FizzBuzz (Modified)

* n = int(input())

for x in range(1, n):

    if x % 2 == 0:

        continue

    elif x % 3 == 0 and x % 5 == 0:

        print("SoloLearn")

    elif x % 3 == 0:

        print("Solo")

    elif x % 5 == 0:

        print("Learn")

    else:

        print(x)

Reusing code

* DRY - Don’t Repeat Yourself
* WET - Write Everything Twice, We Enjoy Typing
* Use functions

Functions

* function\_name(function\_arguments)
* Use def to create functions
* def my\_func():

    print("spam")

    print("spam")

    print("spam")

my\_func()

* Define functions before use
* def welcome\_message():

    name = input()

    print("Welcome,", name)

welcome\_message()

Functions with arguments

* def print\_with\_exclamation(word):

   print(word + "!")

print\_with\_exclamation("spam")

print\_with\_exclamation("eggs")

print\_with\_exclamation("python")

* def print\_sum\_twice(x, y):

   print(x + y)

   print(x + y)

print\_sum\_twice(5, 8)

* o/p:

13

13

* password = input()

repeat = input()

def validate(text1, text2):

    if (text1 == text2):

        print("Correct")

    else:

        print("Wrong")

validate(password, repeat)

* Function arguments or variables can’t be used outside the function
* parameters - variables in function definition
* arguments - values put into parameters when functions are called
* Use return to return a value
* def max(x, y):

    if x >=y:

        return x

    else:

        return y

print(max(4, 7))

z = max(8, 5)

print(z)

* o/p:

7

8

* return can’t be used outside function statement
* Codes after return won’t run
* def add\_numbers(x, y):

    total = x + y

    return total

    print("This won't be printed")

print(add\_numbers(4, 5))

* s = input()

def hashtagGen(text):

    text = text.replace(" ", "")

    text = "#" + text

    return text

print(hashtagGen(s))

* Functions act as variables
* def multiply(x, y):

    return x \* y

a = 4

b = 7

operation = multiply

print(operation(a, b))

* o/p: 28
* Functions act as arguments for other functions
* def add(x, y):

    return x + y

def do\_twice(func, x, y):

    return func(func(x, y), func(x, y))

a = 5

b = 10

print(do\_twice(add, a, b))

* o/p: 30

Modules

* There are 3 types of modules
* User defined, installed from external sources, preinstalled with Python
* Import modules to use them
* import random

for i in range(5):

    value = random.randint(1, 6)

    print(value)

* o/p: 1 to 6 numbers printed randomly
* import random

random.seed(int(input()))

dice1 = random.randint(1,6)

dice2 = random.randint(1,6)

print(dice1)

print(dice2)

* You can import only the certain functions from module
* from math import pi

print(pi)

* Use , to import multiple objects
* from math import pi, sqrt
* import \* imports all objects from module
* from math import \*
* If the module isn’t available, ImportError occurs
* Import module/object under different name using as
* from math import sqrt as square\_root

print(square\_root(100))

* Below code produces an error
* import math as m

print(math.sqrt(25))

Standard library and pip

* Prebuilt Python modules - standard library
* Includes string parsing, data serialization, testing, debugging, manipulating dates, emails, command line arguments, etc

pip

* Many 3rd party Python modules are stored in Python Package Index - PyPI
* Can be installed using pip
* Use pip install library\_name to install

Celsius to Fahrenheit

* celsius = int(input())

def conv(c):

    f = 9/5\*c + 32

    return f

fahrenheit = conv(celsius)

print(fahrenheit)

Exceptions

* Occurs due to incorrect code or input
* ImportError - import failure
* IndexError - list indexing with out of range number
* NameError - unknown variable used
* SyntaxError - code can’t be parsed properly
* TypeError - inappropriate type
* ValueError - inappropriate value
* ZeroDivisionError
* OSError

Exception handling

* Use try/except
* try:

    num1 = 7

    num2 = 0

    print (num1 / num2)

    print("Done calculation")

except ZeroDivisionError:

    print("An error occurred")

    print("due to zero division")

* Can have multiple except blocks
* We can put all exceptions in the same block
* try:

    variable = 10

    print(variable + "hello")

    print(variable / 2)

except ZeroDivisionError:

    print("Divided by zero")

except (ValueError, TypeError):

    print("Error occurred")

* except statement catch all errors
* try:

    word = "spam"

    print(word / 0)

except:

    print("An error occurred")

* Here, ZeroDivisionError occurs, not TypeError
* pin = input()

try:

    int(pin)

    print("PIN code is created")

except ValueError:

    print("Please enter a number")

* def withdraw(amount):

   print(str(amount) + " withdrawn!")

try:

   withdraw(int(input()))

except ValueError:

   print("Please enter a number")

finally

* Use with try/except
* finally block runs no matter what error occurs
* Always runs - even if error isn’t handled by except block
* try:

    print("Hello")

    print(1 / 0)

except ZeroDivisionError:

    print("Divided by zero")

finally:

    print("This code will run no matter what")

* o/p:

Hello

Divided by zero

This code will run no matter what

* coffee = ["Café Latte", "Caffe Americano", "Espresso", "Cappuccino", "Macchiato"]

choice = int(input())

try:

    print(coffee[choice])

except:

    print("Invalid number")

finally:

    print("Have a good day")

else

* else can be used with try/except
* Only runs if no error occurs in try block
* try:

    print(1)

except ZeroDivisionError:

    print(2)

else:

    print(3)

try:

    print(1/0)

except ZeroDivisionError:

    print(4)

else:

    print(5)

* o/p:

1

3

4

* menu = ['Fries', 'Sandwich', 'Cheeseburger', 'Coffee', 'Soda']

try:

    input\_ = int(input())

    print(menu[input\_])

except:

    print("Item not found")

else:

    print("Thanks for your order")

raise: Raising exceptions

* Type of exception must be specified
* print(1)

raise ValueError

print(2)

* try:

print(1/0)

except ZeroDivisionError:

raise ValueError

* o/p: ZeroDivisionError and ValueError
* Exceptions can be raised with arguments
* name = "123"

raise NameError("Invalid name!")

* num = input(“:”)

if float(num) < 0

raise ValueError(“Negative!”)

* Above code raises ValueError exception when the input is negative
* try:

    num = 5 / 0

except:

    print("An error occurred")

    raise

* Above code raises exception occured without arguments
* You can raise outside except block
* You can raise unnamed exception like this
* tweet = input()

try:

    if len(tweet) > 42:

        raise Exception()

except:

    print("Error")

else:

    print("Posted")

* try:

    name = input()

    if len(name) < 4:

        raise Exception()

except:

    print("Invalid Name")

else:

    print("Account Created")

Assertions

* Sanity check that can be turned on or off after testing the program
* If the expression is false exception is raised (AssertionError)
* Use assert
* print(1)

assert 2 + 2 == 4

print(2)

assert 1 + 1 == 3

print(3)

* Generally used in start of function to check valid inputs and after a function to check valid output
* Code doesn’t run after exception is raised
* assert can take second argument
* temp = -10

assert (temp >= 0), "Colder than absolute zero!"

* o/p:

Traceback (most recent call last):

File "file0.py", line 2, in <module>

assert (temp >= 0), "Colder than absolute zero!"

AssertionError: Colder than absolute zero!

* def func(x):

assert x > 0, “Error”

print(x)

Opening files

* If the file is in the same folder
* myfile = open(“filename.txt”)
* If file is in a different folder use the file path
* Use 2nd argument to specify the mode
* r - read mode (default)
* w - write mode (rewriting)
* a - appending
* b - opens in binary mode (for non-text files - images, audio)
* binary mode has to be used with other modes (Ex: wb, rb)
* Ex: wb opens file in binarywrite mode, rb for binaryread mode
* # write mode

open("filename.txt", "w")

# read mode

open("filename.txt", "r")

open("filename.txt")

# binary write mode

open("filename.txt", "wb")

* Use + to give extra access
* Ex: r+ opens file for reading and writing
* Use close to close files
* file.close()

Reading files

* Below code reads everything in the file
* file = open(“sample.txt”, ”r”)

content = file.read()

print(content)

file.close()

* Specify number of bytes to read from a file
* Each ASCII character is 1 byte
* file = open(“sample.txt”, ”r”)

print(file.read(16))

file.close()

* Above code reads 16 bytes from the file
* Now, using read() without arguments will return the rest of the content
* Negative values also will return entire content of the file
* After the file is read, furthur reading will result in empty strings
* Use readlines to return a list containing each line as an element
* file = open(“sample.txt”, “r”)

print(file.readlines())

file.close()

* for loop is used to iterate through lines in file
* file = open(“sample.txt”, “r”)

for line in file:

print(line)

file.close()

* Here, lines are seperated by blank lines since print function automatically adds new line when printing in printline
* readlines put line into a list
* file = open("/usercode/files/pull\_ups.txt")

n = int(input())

list = file.readlines()

print(list[n])

file.close()

Writing files

* Use write
* file = open("newfile.txt", "w")

file.write("This has been written to a file")

file.close()

file = open("newfile.txt", "r")

print(file.read())

file.close()

* o/p: This has been written to a file
* w mode creates a new file if the file doesn’t exist
* In w mode, existing file content is deleted (overwritten)
* file = open("newfile.txt", "w")

file.write("Some new text")

file.close()

file = open("newfile.txt", "r")

print("Reading new contents")

print(file.read())

print("Finished")

file.close()

* if you open a file in w mode, everything inside clears even before .write is used
* Below code returns amount of text written in bytes
* msg = "Hello world!"

file = open("newfile.txt", "w")

amount\_written = file.write(msg)

print(amount\_written)

file.close()

* o/p: 12
* Everything must be converted to string to write
* w+ mode -> Read, Write
* a+ mode -> Read, Write, Append
* names = ["John", "Oscar", "Jacob"]

file = open("names.txt", "w+")

for i in names:

    file.write(i + "\n")

file.close()

file= open("names.txt", "r")

print(file.read())

file.close()

* if writing is successful

file.write(msg) == len(msg)

* To add content to existing file use append mode ‘a’
* file = open("/usercode/files/books.txt", "a")

file.write("\nThe Da Vinci Code")

file.close()

* Close file before opening it again for another mode
* n = int(input())

file = open("numbers.txt", "w+")

for i in range(1, n+1):

    file.write(str(i) + "\n")

file.close()

f = open("numbers.txt", "r")

print(f.read())

f.close()

Working with files

* Make sure files are closed using finally
* This ensures the file is closed even if an error occurs
* try:

f = open(“filename.txt”)

print(f.read())

finally:

f.close()

* Same thing can be done using with
* Temporary variable is created which can only accessed using with statement
* with open(“sample.txt”) as f:

print(f.read())

* Here, file automatically closes after with statement (even if there are exceptions)
* try:

with open(“sample.txt”) as f:

print(f.read())

except:

print(“Error”)

* with open("/usercode/files/books.txt") as f:

   list1 = []

   for i in f.readlines():

      list1.append(i)

   for i in list1:

      print("Line " + str(list1.index(i) + 1) + ": " + str(len(i.split())) + " words")

Quiz

* try:

print(1)

print(20 / 0)

print(2)

except ZeroDivisionError:

print(3)

finally:

print(4)

* o/p:

1

3

4

* try:

print(1)

assert 2 + 2 == 5

except: AssertionError:

print(3)

except:

print(4)

* Here, highest number printed: 3

Book tiles - Quiz

* file = open("/usercode/files/books.txt", "r")

list = file.readlines()

i = 0

for i in range(len(list)):

    item = list[i]

    if item[-1] == "\n":

        print(item[0] + str(len(item)-1))

    else:

        print(item[0] + str(len(item)))

    i += 2

file.close()

* Another way to do this:
* file = open("/usercode/files/books.txt", "r")

list = file.readlines()

str = ""

for i in list:

    for j in (i.split()):

        str += j[0]

    print(str)

    str = ""

file.close()

None

* Represents absenese of a value
* Similar to null in other languages
* Values such as 0, [], “” aren’t equal to None
* functions that has no return, returns None
* def some\_func():

    print("Hi!")

var = some\_func()

print(var)

* print() is equal to None

Dictionaries

* Dictionaries are immutable (can’t be changed)
* Data structures used to map arbitrary keys to values
* Lists can be throught as dictionaries with integer keys within certain range
* Dictionaries can be indexed in the same way as lists, using square brackets
* ages = {"Dave": 24, "Mary": 42, "John": 58}

print(ages["Dave"])

print(ages["Mary"])

* o/p:

24

42

* Indexing a key that isn’t part of the dictionary returns KeyError
* primary = {

    "red": [255, 0, 0],

    "green": [0, 255, 0],

    "blue": [0, 0, 255],

}

print(primary["red"])

print(primary["yellow"])

* Dictionaries can store any type of data as values
* Only immutable objects can be used as keys to dictionaries: strings, booleans, integers, etc
* Immutable: objects that can’t be changed
* Mutable: objects that can be changed, Ex: lists, distionaries
* Using mutable object as a key returns TypeError
* bad\_dict = {

    [1, 2, 3]: "one two three",

}

* Here, TypeError occurs
* Empty dictionary: {}
* store = {"Orange": 2, "Watermelon": 0, "Apple": 8, "Banana": 42}

print(store["Apple"])

* text = input()

dict\_ = {}

count = 0

letter = ""

for i in text:

    dict\_[i] = text.count(i)

print(dict\_)

Dinctionary functions

* Unlike lists new dictionary keys can be assigned values (not just ones that exist)
* squares = {1: 1, 2: 4, 3: "error", 4: 16,}

squares[8] = 64

squares[3] = 9

print(squares)

* o/p: {1: 1, 2: 4, 3: 9, 4: 16, 8: 64}
* Use in and not in to check if a key is in dictionary (just like in a list)
* nums = {

    1: "one",

    2: "two",

    3: "three",

}

print(1 in nums)

print("three" in nums)

print(4 not in nums)

* o/p:

True

False

True

* Use get (similar to indexing) to return a value using key
* If key isn’t found specified value returns instead of KeyError (Default - None)
* pairs = {1: "apple",

    "orange": [2, 3, 4],

    True: False,

    None: "True",

}

print(pairs.get("orange"))

print(pairs.get(7))

print(pairs.get(12345, "not in dictionary"))

* o/p:

[2, 3, 4]

None

not in dictionary

* books = {

    "Life of Pi": "Adventure Fiction",

    "The Three Musketeers": "Historical Adventure",

    "Watchmen": "Comics",

    "Bird Box": "Horror",

    "Harry Potter":"Fantasy Fiction",

    "Good Omens": "Comedy"

}

book = input()

print(books.get(book, "Book not found"))

* nums = {1:1, 2:1, 3:2, 4:3}

print(nums.get(4,0) + nums.get(7,5))

* o/p: 8
* contacts = {

    "David": ["123-321-88", "david@test.com"],

    "James": ["241-879-093", "james@test.com"],

    "Bob": ["987-004-322", "bob@test.com"],

    "Amy": ["340-999-213", "a@test.com"]

}

name = input()

if name in contacts:

    print(contacts[name][1])

else:

    print("Not found")

* Sololearn: Revenue growth analysis

data = {

    "100-90": 25, "42-01": 48, "55-09": 12, "128-64": 71, "002-22": 18, "321-54": 19, "097-32": 33, "065-135": 64, "99-043": 80, "111-99": 11, "123-019": 5, "109-890": 72, "132-123": 27, "32-908": 27, "008-09": 25, "055-967": 35, "897-99": 44, "890-98": 56, "344-32": 65, "43-955": 59, "001-233": 9, "089-111": 15, "090-090": 17, "56-777": 23, "44-909": 27, "13-111": 21, "87-432": 15, "87-433": 14, "87-434": 23, "87-435": 11, "87-436": 12, "87-437": 16, "94-121": 15, "94-122": 35, "80-089": 10, "87-456": 8, "87-430": 40

}

age = int(input())

new = 0

old = 0

for i in data.values():

    if i < age:

        new += 5

    else:

        new += 20

for i in data.values():

    if i < 18:

        old += 5

    else:

        old += 20

print(int((new - old)/old\*100))

Tuples

* Similar to lists but immutable (can’t be changed)
* words = (“spam”, “eggs”, “sausages”)
* Tuples have indexes just like lists
* words = ("spam", "eggs", "sausages",)

print(words[0])

* o/p: spam
* Since tuples are immutable, reassigning values will cause TypeError
* words = ("spam", "eggs", "sausages",)

words[1] = "cheese"

* o/p: TypeError
* Tuples can be nested with each other (just like lists and dictionaries)
* You can create tuples without parantheses
* my\_tuple = "one", "two", "three"

print(my\_tuple[0])

* o/p: one
* Empty tuple: tpl = ()
* Tuples are faster than lists
* import math

p1 = (23, -88)

p2 = (6, 42)

distance = math.sqrt((p1[0]-p2[0])\*\*2 + (p1[1]-p2[1])\*\*2)

print(distance)

* hash(tuple) returns hash value of tuple
* n = int(input())

inputs = input().split()

for i in range(len(inputs)):

    inputs[i] = int(inputs[i])

t = tuple(inputs)

print(hash(t))

Tuple unpacking

* numbers = (1, 2, 3)

a, b, c = numbers

print(a)

print(b)

print(c)

* o/p:

1

2

3

* Use \* to take all values that’s left over
* a, b, \*c, d = [1, 2, 3, 4, 5, 6, 7, 8, 9]

print(a)

print(b)

print(c)

print(d)

* o/p:

1

2

[3, 4, 5, 6, 7, 8]

9

* def calc(x):

    p = 4\*x

    a = x\*x

    return p, a

side = int(input())

p, a = calc(side)

print("Perimeter: " + str(p))

print("Area: " + str(a))

* points = [

    (12, 55),

    (880, 123),

    (64, 64),

    (190, 1024),

    (77, 33),

    (42, 11),

    (0, 90)

]

min\_distance = []

for i in points:

    x, y = i

    distance = (x\*\*2 + y\*\*2)\*\*0.5

    min\_distance.append(distance)

print(min(min\_distance))

List slices

* Advanced way of retrieving values from a list
* Here, 2nd index isn’t included just like in range
* squares = [0, 1, 4, 9, 16, 25, 36, 49, 64, 81]

print(squares[2:6])

print(squares[3:8])

print(squares[0:1])

* o/p:

[4, 9, 16, 25]

[9, 16, 25, 36, 49]

[0]

* if 1st number is omitted - Taken as start of the list
* if 2nd number is omitted - Taken as end of the list
* squares = [0, 1, 4, 9, 16, 25, 36, 49, 64, 81]

print(squares[:7])

print(squares[7:])

* o/p:

[0, 1, 4, 9, 16, 25, 36]

[49, 64, 81]

* Tuples also can be sliced
* List slices can have 3rd number for step
* squares = [0, 1, 4, 9, 16, 25, 36, 49, 64, 81]

print(squares[::2])

print(squares[2:8:3])

* o/p:

[0, 4, 16, 36, 64]

[4, 25]

* Negative values can be used in slicing
* squares = [0, 1, 4, 9, 16, 25, 36, 49, 64, 81]

print(squares[1:-1])

* o/p: [1, 4, 9, 16, 25, 36, 49, 64]
* Slice is going backwards when the 3rd number is negative
* x = input()

elements = x.split()

print(elements[-1])

* sqs = [0, 1, 4, 9, 16, 25, 36, 49, 64, 81]

print(sqs[7:5:-1])

* o/p: [49, 36]

List comprehensions

* Useful way of quickly creating lists
* cubes = [i\*\*3 for i in range(5)]

print(cubes)

* o/p: [0, 1, 8, 27, 64]
* Inspired by set-builder notation in maths
* Use if with list comprehensions
* evens=[i\*\*2 for i in range(10) if i\*\*2 % 2 == 0]

print(evens)

* o/p: [0, 4, 16, 36, 64]
* Creating list in extensive range will return MemoryError
* even = [2\*i for i in range(10\*\*100)]
* This can be solved using generators
* x = int(input())

list = [i for i in range(x) if i%3 == 0 and i%5 == 0]

print(list)

* x = int(input())

y = int(input())

z = int(input())

n = int(input())

listA = [[i, j, k] for i in range(x + 1) for j in range(y + 1) for k in range(z + 1) if i + j + k != n]

print(listA)

* word = input()

list\_ = ["a", "e", "i", "o", "u"]

out = [i for i in word if i not in list\_]

print(out)

String formatting

* nums = [4, 5, 6]

msg = "Numbers: {0} {1} {2}". format(nums[0], nums[1], nums[2])

print(msg)

* o/p: Numbers: 4 5 6
* Commas in format method doesn’t create spaces
* print(“{0}{1}{0}”.format(“abra”,”cad”))
* o/p: abracadabra
* Named arguments can be used too
* a = "{x}, {y}".format(x=5, y=12)

print(a)

* o/p: 5 12
* name = input()

age = int(input())

print("{} is {} years old".format(name,age))

String functions

* join - join list of strings with another string as seperator
* print(", ".join(["spam", "eggs", "ham"]))
* o/p: spam, eggs, ham
* replace - replaces one substring in a string with another
* print("Hello ME".replace("ME", "world"))
* o/p: Hello world
* txt = input()

print(txt.replace("#"," "))

* startswith and endswith - determine if there’s a substring at the start and end of string
* print("This is a sentence.".startswith("This"))

print("This is a sentence.".endswith("sentence."))

* o/p:

True

True

* lower and upper - change the case of string
* print("This is a sentence.".upper())

print("AN ALL CAPS SENTENCE".lower())

* o/p:

THIS IS A SENTENCE.

an all caps sentence

* split - opposite of join
* print("spam, eggs, ham".split(", "))
* o/p: ['spam', 'eggs', 'ham']
* txt = input()

list = txt.split(" ")

print(len(list))

* HackerRank: Alphabet Rangoli

**def** print\_rangoli(size):

    alp = " abcdefghijklmnopqrstuvwxyz"

**for** i **in** **range**(size, 0, -1):

        c = alp[size:i:-1] + alp[i:size+1]

        c = "-".join(c)

**print**(c.center((size\*4)-3, "-"))

**for** i **in** **range**(size-1):

        c = alp[size:i+2:-1] + alp[i+2:size+1]

        c = "-".join(c)

**print**(c.center((size\*4)-3, "-"))

**if** \_\_name\_\_ == '\_\_main\_\_':

    n = **int**(**input**())

    print\_rangoli(n)

* HackerRank: Capitalize!

*#!/bin/python3*

**import** math

**import** os

**import** random

**import** re

**import** sys

**def** solve(s):

**return** " ".join([i.capitalize() **for** i **in** s.split(" ")])

**if** \_\_name\_\_ == '\_\_main\_\_':

    fptr = **open**(os.environ['OUTPUT\_PATH'], 'w')

    s = **input**()

    result = solve(s)

    fptr.write(result + '\n')

    fptr.close()

Numeric functions

* max - Maximum of numbers on a list
* min - Minimum of numbers on a list
* print(min(1, 2, 3, 4, 0, 2, 1))

print(max([1, 4, 9, 2, 5, 6, 8]))

* o/p:

0

9

* abs - Absolute value
* print(abs(-99))

print(abs(42))

* o/p:

99

42

* nums = [-1,2,-3,4,-5]

if all([abs(i) < 3 for i in nums]):

print(1)

else:

print(2)

* o/p: 2
* round - To round a number
* print(round(15.5))
* o/p: 16
* sum - Total
* print(sum([1, 2, 3, 4, 5]))
* o/p: 15

Text analyzer - Program

* Below code read content of file
* filename = input("Enter a filename: ")

with open(filename) as f:

text = f.read()

print(text)

* Below function counts how many times a character occurs in a string
* def count\_char(text, char):

count = 0

for c in text:

if c == char:

count += 1

return count

filename = input("Enter a filename: ")

with open(filename) as f:

text = f.read()

print(count\_char(text, "r"))

* o/p:

Enter a filename: sample.txt

83

* Below code finds percentage of text each character in alphabet occupies
* for char in "abcdefghijklmnopqrstuvwxyz":

perc = 100 \* count\_char(text, char) / len(text)

print("{0} - {1}%".format(char, round(perc, 2)))

* Full code:

def count\_char(text, char):

    count = 0

    for c in text:

        if c == char:

            count += 1

    return count

file = open("newfile.txt", "w")

file.write("""Ornhgvshy vf orggre guna htyl.

Rkcyvpvg vf orggre guna vzcyvpvg.

Fvzcyr vf orggre guna pbzcyvpngrq.

Syng vf orggre guna arfgrq.

Fcenfr fv orggre guna qrafr.

Ernqnovyvgl pbhagf.

Fcrpvny pnfrf nera'g fcrpvny rabthu gb oernx gur ehyrf.

Nygubhtu cenpgvpnyvgl orgnf chevgl.

Reebef fubhyq arire cnff fvyragyl.

Hayrff rkcyvpvgyl fvyraprq.

Va gur snpr bs nzovthvgl, ershfr gur grzcgngvba bg thrff.

Gurer fubhyq or bar-- naq cersrenoylbayl bar --boivbhf jnl gb qb vg.

Nygubhtu gung jnl znl abg or boivbhf ng svefg hayrff lbh'er Qhgpu.

Abj vf orggre guna arrire.

Nygubhtu arire vf bsgra orggre guna \*evtug\* abj.

Vs gur vzcyrzragngvba vf uneq gb rkcynva, vg'f n onq vqrn.

Vs gur vzcyrzragngvba vf rnfl gb rkcynva, vg znl or n tbbq vqrn.

Anzrfcnprf ner bar ubaxvat terng vqrn -- yrg'f qb zber bs gubfr!""")

file.close()

filename = "newfile.txt"

with open(filename) as f:

    text = f.read()

for char in "abcdefghijklmnopqrstuvwxyz":

    perc = 100 \* count\_char(text, char) / len(text)

    print("{0} - {1}%".format(char, round(perc, 2)))

* o/p:

a - 4.68%

b - 4.94%

c - 2.28%

d - 0.0%

e - 3.8%

f - 5.19%

g - 8.99%

h - 2.53%

i - 0.63%

j - 0.51%

k - 0.51%

l - 1.9%

m - 0.0%

n - 6.2%

o - 2.28%

p - 1.9%

q - 2.03%

r - 10.51%

s - 1.27%

t - 1.39%

u - 3.54%

v - 6.08%

w - 0.0%

x - 0.25%

y - 3.92%

z - 1.65%

Quiz

* txt = input()

list = txt.split(" ")

long = ""

for i in list:

    if len(i) > len(long):

        long = i

print(long)

User defined data structures

* Stacks, queues, linked lists, graphs

Stack

* simple data structure that adds and removes elements in a particular order
* LIFO: Last In, First Out
* Only the last added element can be removed
* Push: adding new element onto stack
* Pop: removinf an element from stack
* Can be used to create undo-redo functions, parsing expressions (infix to postfix/prefix conversion), etc
* Stack can be implemented using list
* class Stack:

    def \_\_init\_\_(self):

        self.items = []

    def is\_empty(self):

        return self.items == []

    def push(self, item):

        self.items.insert(0, item)

    def pop(self):

        return self.items.pop(0)

    def print\_stack(self):

        print(self.items)

s = Stack()

s.push('a')

s.push('b')

s.push('c')

s.print\_stack()

s.pop()

s.print\_stack()

* o/p:

['c', 'b', 'a']

['b', 'a']

* class Browser:

    def \_\_init\_\_(self):

      self.links = []

    def is\_empty(self):

      return self.links == []

    def push(self, link):

      self.links.insert(0, link)

    def pop(self):

      return self.links.pop(0)

x = Browser()

x.push('about:blank')

x.push('www.sololearn.com')

x.push('www.sololearn.com/courses/')

x.push('www.sololearn.com/courses/python/')

while not x.is\_empty():

    print(x.pop())

* Sololearn: Balanced paranthese

class Stack:

    def \_\_init\_\_(self):

        self.items = []

    def is\_empty(self):

        return self.items == []

    def push(self, item):

        self.items.insert(0, item)

    def pop(self):

        return self.items.pop(0)

    def print\_stack(self):

        return self.items

def balanced(expression):

    s = Stack()

    for i in expression:

        if i == "(":

            s.push(i)

        if i == ")":

            if "(" in s.print\_stack():

                s.pop()

            else:

                return False

    if not s.print\_stack():

        return True

    else:

        return False

print(balanced(input()))

Queue

* Similar to stack, but different way of adding and removing elements
* FIFO: First In First Out
* Inserted from rear (enqueue) and deleted from front (dequeue)
* Applications: printing documents on printer, call center systems answering people on hold, etc
* class Queue:

    def \_\_init\_\_(self):

        self.items = []

    def is\_empty(self):

        return self.items == []

    def enqueue(self, item):

        self.items.insert(0, item)

    def dequeue(self):

        return self.items.pop()

    def print\_queue(self):

        print(self.items)

q = Queue()

q.enqueue('a')

q.enqueue('b')

q.enqueue('42')

q.print\_queue()

q.dequeue()

q.print\_queue()

* o/p: ['42', 'b', 'a']

['42', 'b']

* Here, pop() is equal to pop(-1)
* Sololearn: callcenter application

class CallCenter:

    def \_\_init\_\_(self):

      self.customers = []

    def is\_empty(self):

      return self.customers == []

    def add(self, x):

      self.customers.insert(0, x)

    def next(self):

      return self.customers.pop()

c = CallCenter()

while True:

    n = input()

    if n == 'end':

        break

    c.add(n)

time = 0

while True:

  if c.is\_empty():

    break

  item = c.next()

  if item == 'general':

    time += 5

  elif item == 'technical':

    time += 10

print(time)

Linked list

* Sequence of nodes where each node stores its own data and a link to next node
* First node: head: Starting point for any iteration through the list
* Last: node points to None: To determine the end of list
* You can insert and remove nodes in any position of the linked list (similar to standard list)
* Applications: Undo/redo functionality, music playlist
* Linked lists are used to create other data structures like stacks, queues and graphs
* class Node:

    def \_\_init\_\_(self, data, next):

        self.data = data

        self.next = next

class LinkedList:

    def \_\_init\_\_(self):

        self.head = None

    def add\_at\_front(self, data):

        self.head = Node(data, self.head)

    def add\_at\_end(self, data):

        if not self.head:

            self.head = Node(data, None)

            return

        curr = self.head

        while curr.next:

            curr = curr.next

        curr.next = Node(data, None)

    def get\_last\_node(self):

        n = self.head

        while(n.next != None):

            n = n.next

        return n.data

    def is\_empty(self):

        return self.head == None

    def print\_list(self):

        n = self.head

        while n != None:

            print(n.data, end = " => ")

            n = n.next

        print()

s = LinkedList()

s.add\_at\_front(5)

s.add\_at\_end(8)

s.add\_at\_front(9)

s.print\_list()

print(s.get\_last\_node())

* o/p:

9 => 5 => 8 => 8

* Here, add\_at\_front() adds a new node as the head of the list and links the previous head to it
* add\_at\_end() iterates to the end of the list using while loop and adds a new node as the link of the last node
* class Track:

    def \_\_init\_\_(self, title, next):

        self.title = title

        self.next = next

class Player:

    def \_\_init\_\_(self):

        self.head = None

    def add(self, title):

        if not self.head:

            self.head = Track(title, None)

            return

        curr = self.head

        while curr.next:

            curr = curr.next

        curr.next = Track(title, None)

p = Player()

while True:

    x = input()

    if x == 'end':

        break

    p.add(x)

n = p.head

while n != None:

    print(n.title)

    n = n.next

Graph

* Applications: Networks, transportation paths of a city, social network connections
* Graph: Set of connected notes
* Node: vertex
* Connection: edge
* Graph can be represented using a square matrix
* Element: edge
* Ex: 0 means no edge, 1 means edge
* Rows and columns: vertices
* 0 1 1

1 0 0

1 0 0

* Here, matrix has 3 vertices: 3x3 matrix
* 1st vertex is connected to 2nd and 3rd
* This type of matrix: adjacency matrix: shows if the corresponding vertices are adjacent or not
* class Graph():

    def \_\_init\_\_(self, size):

        self.adj = [ [0] \* size for i in range(size)]

        self.size = size

    def add\_edge(self, orig, dest):

        if orig > self.size or dest > self.size or orig < 0 or dest < 0:

            print("Invalid Edge")

        else:

            self.adj[orig-1][dest-1] = 1

            self.adj[dest-1][orig-1] = 1

    def remove\_edge(self, orig, dest):

        if orig > self.size or dest > self.size or orig < 0 or dest < 0:

            print("Invalid Edge")

        else:

            self.adj[orig-1][dest-1] = 0

            self.adj[dest-1][orig-1] = 0

    def display(self):

        for row in self.adj:

            print()

            for val in row:

                print('{:4}'.format(val),end="")

#a sample Graph

G = Graph(4)

G.add\_edge(1, 3)

G.add\_edge(3, 4)

G.add\_edge(2, 4)

G.display()

* o/p:

0 0 1 0

0 0 0 1

1 0 0 1

0 1 1 0

* Here, adj is the list that contains 2D matrix
* \_\_init\_\_ method creates adj matrix with the given size (number of vertices) and initializes all values to zeros
* add\_edge() method is used to add an edge by setting the corresponding values 1
* remove\_edge() sets the values to 0
* Sololearn: Social network connections

class X():

    def \_\_init\_\_(self, size):

        self.adj = [ [0] \* size for i in range(size)]

        self.size = size

    def add\_friend(self, x, y):

        if x > self.size or y > self.size or x < 0 or y < 0:

            print("Error")

        else:

            self.adj[x-1][y-1] = 1

            self.adj[y-1][x-1] = 1

    def remove\_friend(self, x, y):

        if x > self.size or y > self.size or x < 0 or y < 0:

            print("Error")

        else:

            self.adj[x-1][y-1] = 0

            self.adj[y-1][x-1] = 0

x = X(5)

x.add\_friend(1, 3)

x.add\_friend(1, 5)

x.add\_friend(2, 5)

x.add\_friend(2, 4)

x.add\_friend(4, 5)

n = int(input())

print(x.adj[n - 1][0] + x.adj[n - 1][1] + x.adj[n - 1][2] + x.adj[n - 1][3] + x.adj[n - 1][4])

* In adjacency matrix, m[x][y] == m[y][x]

Functional programming

* Style of programming based on functions
* higher order functions - Functions taking other functions as arguments
* def apply\_twice(func, arg):

    return func(func(arg))

def add\_five(x):

    return x + 5

print(apply\_twice(add\_five, 10))

* o/p: 20
* def test(func,arg):

return func(func(arg))

def mult(x)

return x\*x

print(test(mult,2))

* o/p: 16

Pure functions

* Functional programming seek to use pure functions
* Below is a pure function
* def pure\_func(x,y):

temp = x + 2\*y

return temp/(2\*x + y)

* Below is an impure function. Impure functions have side effects
* some\_list = []

def impure\_func(arg):

some\_list.append(arg)

* memorization - function input is stored to use again for next time function input is needed (increases efficiency)

Lambdas

* Functions created using def assigns it to a variable automatically
* To create functions without assigning to variable use lambda
* These functions are called anonymous
* Mainly used in functional programming
* def func(f,arg):

return f(arg)

func(lambda x: 2\*x\*x, 5)

* Since lambdas can have only single expression they can’t do much like functions
* print((lambda x: x\*\*2 + 5\*x + 4) (-4))
* o/p: 0
* Defined function for above code is as follows
* def polynomial(x):

    return x\*\*2 + 5\*x + 4

print(polynomial(-4))

* o/p: 0
* Lambda functions can be assigned to variables like normal functions
* double = lambda x: x\*2

print(double(7))

* o/p: 14
* x = int(input())

y = (lambda z:z\*\*3)(x)

print(y)

* triple = lambda x: x\*3

add = lambda x,y: x + y

print(add(triple(3),4))

* o/p: 13
* price = int(input())

perc = int(input())

res = (lambda x,y:x\*y/100)(price, perc)

print(res)

map

* map and filter are built in functions
* map - takes a function and iterable as arguments and returns a new iterable with function applied to each argument
* def add\_five(x):

    return x + 5

nums = [11, 22, 33, 44, 55]

result = list(map(add\_five, nums))

print(result)

* o/p: [16, 27, 38, 49, 60]
* We use list() to convert that to a list
* Below code returns the same answer using lambda
* nums = [11, 22, 33, 44, 55]

result = list(map(lambda x: x + 5, nums))

print(result)

* o/p: [16, 27, 38, 49, 60]
* salaries = [2000, 1800, 3100, 4400, 1500]

bonus = int(input())

salaries = list(map(lambda x: x + bonus, salaries))

print(salaries)

filter

* Filters an iterable by removing items that don’t match a predicate (condition: function that returns a Boolean)
* nums = [11, 22, 33, 44, 55]

res = list(filter(lambda x: x % 2==0, nums))

print(res)

* o/p: [22, 44]
* Just like map, use a list to print the output
* names = ["David", "John", "Annabelle", "Johnathan", "Veronica"]

ans = list(filter(lambda x: len(x) > 5, names))

print(ans)

Generators

* Type of iterable like lists or tuples
* They don’t allow arbitrary indices but can be iterated with for loops
* Use yield and functions to create
* def countdown():

    i=5

    while i > 0:

        yield i

        i -= 1

for i in countdown():

    print(i)

* o/p:

5

4

3

2

1

* Yield statement is used to define a generator
* Yield is used to replace the return of a function to provide a result to its caller withot destroying local variables
* Generators don’t have memory restrictions like lists - infinite memory: since they return 1 item at a time
* def infinite\_sevens():

while True:

yield 7

for i in infinite\_sevens():

print(i)

* o/p: Code runs infinitely printing 7
* Generators allow to declare a function that behaves like an iterator that can be used in a for loop
* Finite generators can be converted into lists
* def numbers(x):

    for i in range(x):

        if i % 2 == 0:

            yield i

print(list(numbers(11)))

* o/p: [0, 2, 4, 6, 8, 10]
* Generators improves performance: lower memory usage
* txt = input()

def words():

    for i in txt.split(" "):

        yield i

print(list(words()))

* def make\_word():

word = “”

for ch in “spam”:

word += ch

yield word

print(list(make\_word()))

* o/p: [‘s’, ’sp’, ‘spa’, ‘spam’]
* def isPrime(x):

    if x < 2:

        return False

    elif x == 2:

        return True

    for n in range(2, x):

        if x % n ==0:

            return False

    return True

def primeGenerator(a, b):

    for i in range(a, b):

        if isPrime(i) == True:

            yield i

f = int(input())

t = int(input())

print(list(primeGenerator(f, t)))

Decorators

* Provides a way to modify functions using other functions
* def decor(func):

  def wrap():

    print("============")

    func()

    print("============")

  return wrap

def print\_text():

  print("Hello world!")

decorated = decor(print\_text)

decorated()

* o/p:

============

Hello world!

============

* def decor(func):

    def wrap():

        print("============")

        func()

        print("============")

    return wrap

def print\_text():

    print("Hello world!")

print\_text = decor(print\_text)

print\_text()

* Now print\_text corresponds to our decorated version
* We can decorate function using @ and decorater name
* def decor(func):

    def wrap():

        print("============")

        func()

        print("============")

    return wrap

@decor

def print\_text():

    print("Hello world!")

print\_text()

* o/p:

============

Hello world!

============

* Single function can have multiple decorators
* text = input()

def uppercase\_decorator(func):

    def wrapper(text):

        return func(text).upper()

    return wrapper

@uppercase\_decorator

def display\_text(text):

    return(text)

print(display\_text(text))

* @dec is same as func = dec(func)
* def decor(func):

    def wrap(num):

        print("\*\*\*")

        func(num)

        print("\*\*\*\nEND OF PAGE")

    return wrap

@decor

def invoice(num):

    print("INVOICE #" + num)

invoice(input())

Recursion

* def factorial(x):

    if x == 1:

        return 1

    else:

        return x \* factorial(x-1)

print(factorial(5))

* o/p: 120
* 1! is the base case which acts as the exit condition of the recursion
* def sum\_to(x):

return x + sum\_to(x - 1)

print(sum\_to(5))

* o/p: RuntimeError
* Recursion can also be indirect
* def is\_even(x):

    if x == 0:

        return True

    else:

        return is\_odd(x-1)

def is\_odd(x):

    return not is\_even(x)

print(is\_odd(17))

print(is\_even(23))

* o/p:

True

False

* def calc(list):

    if len(list)==0:

        return 0

    else:

        return list[0]\*\*2 + calc(list[1:])

list = [1, 3, 4, 2, 5]

x = calc(list)

print(x)

* o/p: 55
* def fib(x):

if x == 0 or x == 1:

return 1

else:

return fib(x-1) + fib(x-2)

print(fib(4))

* o/p: 5
* Below code is a binary converter
* def convert(num):

   if num == 0:

      return 0

   else:

      return (num % 2 + 10 \* convert(num // 2))

print(convert(int(input())))

* def spell(txt):

    if txt=="":

        return txt

    else:

        print(txt[len(txt)-1])

        return spell(txt[0:len(txt)-1])

txt = input()

print(spell(txt))

\*args

* Can have multiple number of arguments
* \*args must be used after named parameters
* def function(named\_arg, \*args):

    print(named\_arg)

    print(args)

function(1, 2, 3, 4, 5)

* o/p:

1

(2, 3, 4, 5)

* Here, args are accessible as tuple args
* def my\_min(\*args):

    return(min(args))

print(my\_min(8, 13, 4, 42, 120, 7))

\*\*kwargs

* kwargs: keyword arguments
* Allows to handle named arguments that haven’t defined in advance
* Returns a dictionary: Keys are argument names and values are argument values
* def my\_func(x, y=7, \*args, \*\*kwargs):

    print(kwargs)

my\_func(2, 3, 4, 5, 6, a=7, b=8)

* o/p:

{'a': 7, 'b': 8}

* \*\*kwargs aren’t inclued in \*args

Sets

* Data structures similar to lists or dictionaries
* Use curly braces
* num\_set = {1, 2, 3, 4, 5}

word\_set = set(["spam", "eggs", "sausage"])

print(3 in num\_set)

print("spam" not in word\_set)

* o/p:

True

False

* Sets use in and len
* Since they are unordered, they can’t be indexed
* Can’t contain duplicates
* Use add instead of append
* remove removes specific element while pop removes arbitrary element
* nums = {1, 2, 1, 3, 1, 4, 5, 6}

print(nums)

nums.add(-7)

nums.remove(3)

print(nums)

* o/p:

{1, 2, 3, 4, 5, 6}

{1, 2, 4, 5, 6, -7}

* Sets can be combined using mathematical operations
* Union operator | combines 2 sets
* Intersection operator &
* Difference operator - gets items in the first but not in the second
* Symmetric difference operator ^ gets items in either set but not both
* first = {1, 2, 3, 4, 5, 6}

second = {4, 5, 6, 7, 8, 9}

print(first | second)

print(first & second)

print(first - second)

print(second - first)

print(first ^ second)

* o/p:

{1, 2, 3, 4, 5, 6, 7, 8, 9}

{4, 5, 6}

{1, 2, 3}3

{8, 9, 7}

{1, 2, 3, 7, 8, 9}

* skills = {'Python', 'HTML', 'SQL', 'C++', 'Java', 'Scala'}

job\_skills = {'HTML', 'CSS', 'JS', 'C#', 'NodeJS'}

print(list(skills & job\_skills)[0])

* s1 = input()

s2 = input()

s1 = set(s1.split())

s2 = set(s2.split())

count = 0

for i in s1:

    if i in s2:

        count += 1

print(count)

itertools

* count function counts up infinitely from a value
* cycle function iterates through an iterable (list, string, etc)
* repeat function repeats an object
* from itertools import count

for i in count(3):

    print(i)

    if i >=11:

        break

* o/p:

3

4

5

6

7

8

9

10

11

* Some functions in itertools operate on iterables similar way to map and filter
* takewhile - takes items from an iterable while a predicate function remains true
* chain - combines several iterables into one long one
* accumulate - returns a running total of values in an iterable
* from itertools import accumulate, takewhile

nums = list(accumulate(range(8)))

print(nums)

print(list(takewhile(lambda x: x<= 6, nums)))

* o/p:

[0, 1, 3, 6, 10, 15, 21, 28]

[0, 1, 3, 6]

* from itertools import takewhile

nums = [2,4,6,7,9,8]

a = takewhile(lambda x: x%2 == 0, nums)

print(list(a))

* Takewhile is similar to filter
* Combinatoric functions in itertool - product and permuatation
* from itertools import product, permutations

letters = ("A", "B")

print(list(product(letters, range(2))))

print(list(permutations(letters)))

* o/p:

[('A', 0), ('A', 1), ('B', 0), ('B', 1)]

[('A', 'B'), ('B', 'A')]

* from itertools import permutations

items = ['x', 'y']

print(list(permutations(items)))

Quiz - Module 7

* Anonymous function and call it for number 6
* a = (lambda x: x\*(x + 1))(6)

print(a)

* To leave only even numbers
* nums = [1, 2, 8, 3, 7]

res = list(filter(lambda x: x%2 == 0, nums))

print(res)

Fibonacci - Project

* num = int(input())

def fibonacci(n):

    if n == 1 or n == 0:

        return n

    else:

        return fibonacci(n - 1) + fibonacci(n - 2)

for i in range(num):

    print(fibonacci(i))

Classes

* 3 paradigms of programming - imperative (statements, loops, functions as subroutines), functional (pure functions, higher order functions, recursion) and OOP
* Use class to create classes
* class methods are functions
* class Cat:

def \_\_init\_\_(self, color, legs):

self.color = color

self.legs = legs

felix = Cat(“ginger”, 4)

rover = Cat(“dog-colored”, 4)

stumpy = Cat(“brown”, 3)

* Here, color and legs are attribute
* Cat is the class
* self, color and legs are objects (instants) of that class

\_\_init\_\_

* \_\_init\_\_ is a method: Most important method in a class: constructor
* methods should have self: Instance calling the method
* self must be the first argument
* class Cat:

    def \_\_init\_\_(self, color, legs):

        self.color = color

        self.legs = legs

felix = Cat("ginger", 4)

print(felix.color)

* o/p: ginger
* class Student:

    def \_\_init\_\_(self, name):

        self.name = name

    def greet(self):

        print(self.name+" says hi")

obj = Student("John")

obj.greet()

Methods

* All methods have self as the 1st parameter
* class Dog:

  def \_\_init\_\_(self, name, color):

    self.name = name

    self.color = color

  def bark(self):

    print("Woof!")

fido = Dog("Fido", "brown")

print(fido.name)

fido.bark()

* o/p:

Fido

Woof!

* Classes can have class attributes
* They are shared by all instances (objects) of the class
* class Dog:

    legs = 4

    def \_\_init\_\_(self, name, color):

        self.name = name

        self.color = color

fido = Dog("Fido", "brown")

print(fido.legs)

print(Dog.legs)

* o/p:

4

4

* Trying to access attribute of instance not defined returns AttributeError
* class Player:

    def \_\_init\_\_(self, name, level):

        self.name = name

        self.level = level

    def intro(self):

        print(self.name + " (Level " + self.level + ")")

player = Player(input(), input())

player.intro()

OOP

* Programming model that organizes software design aroud data or objects rather than functions and logic
* Inheritance - Inheriting attributes and methods of another class
* Encapsulation - Hides data from outside access
* Polymorphism - Function having same name but carrying different functionalities
* Data abstraction - Hiding internal details of a function and showing its functionality only

Inheritance

* Allows to share functionality between classes
* To inherit from another class put superclass name in parantheses
* class Animal:

    def \_\_init\_\_(self, name, color):

        self.name = name

        self.color = color

class Cat(Animal):

    def purr(self):

        print("Purr...")

class Dog(Animal):

    def bark(self):

        print("Woof!")

fido = Dog("Fido", "brown")

print(fido.color)

fido.bark()

* o/p:

brown

Woof!

* You can override attributes from superclass
* class Wolf:

    def \_\_init\_\_(self, name, color):

        self.name = name

        self.color = color

    def bark(self):

        print("Grr...")

class Dog(Wolf):

    def bark(self):

        print("Woof")

husky = Dog("Max", "grey")

husky.bark()

* o/p: Woof
* class Vehicle:

    def horn(self):

        print("Beep!")

class Car(Vehicle):

    def \_\_init\_\_(self, name, color):

        self.name = name

        self.color = color

obj = Car("BMW", "red")

obj.horn()

* Inheritance can be indirect
* One class can inherit from another which is also inherited
* class A:

    def method(self):

        print("A method")

class B(A):

    def another\_method(self):

        print("B method")

class C(B):

    def third\_method(self):

        print("C method")

c = C()

c.method()

c.another\_method()

c.third\_method()

* o/p:

A method

B method

C method

* However, circular inheritance isn’t possible
* class A:

def a(self):

print(1)

class B(A):

def a(self):

print(2)

class C(B):

def c(self):

print(3)

c = C()

c.a()

* o/p: 2
* Here, superclass of class C is class B
* super function refers to the parent class
* class A:

    def spam(self):

        print(1)

class B(A):

    def spam(self):

        print(2)

        super().spam()

B().spam()

* o/p:

2

1

* Here, super().spam() calls the spam method of superclass
* class Shape:

    def \_\_init\_\_(self, w, h):

        self.width = w

        self.height = h

    def area(self):

        print(self.width\*self.height)

class Rectangle(Shape):

    def perimeter(self):

        print(2\*(self.width + self.height))

w = int(input())

h = int(input())

r = Rectangle(w, h)

r.area()

r.perimeter()

Magic methods

* Magic methods have double underscores \_\_ at the beginning and end of their names
* Also called dunders
* Ex: \_\_init\_\_, \_\_add\_\_
* \_\_init\_\_ is the magic method for creating an instance
* One common use: operator overloading
* + and \* can be used
* \_\_add\_\_ for +
* class Vector2D:

    def \_\_init\_\_(self, x, y):

        self.x = x

        self.y = y

    def \_\_add\_\_(self, other):

        return Vector2D(self.x + other.x, self.y + other.y)

first = Vector2D(5, 7)

second = Vector2D(3, 9)

result = first + second

print(result.x)

print(result.y)

* o/p:

8

16

* \_\_sub\_\_ for -
* \_\_mul\_\_ for \*
* \_\_truediv\_\_ for i
* \_\_floordiv\_\_ for //
* \_\_mod\_\_ for %
* \_\_pow\_\_ for \*\*
* \_\_and\_\_ for &
* \_\_xor\_\_ for ^
* \_\_or\_\_ for |
* Here, \_\_add\_\_ allows a custom behaviour for + operator in class
* The expression first + second is translated into first.\_\_add\_\_(second)
* If x isn’t implemented \_\_add\_\_ and first and second are of different types, second.\_\_radd\_\_(first) is called
* There are r methods for all magic methods
* Ex: A()^B() is evaluated as B().\_\_rxor\_\_(A()) if A doesn’t implement any magic methods
* Once defined you can add 2 objects of the class together
* class BankAccount:

    def \_\_init\_\_(self, balance):

        self.balance = balance

    def \_\_add\_\_(self, other):

        return BankAccount(self.balance + other.balance)

a = BankAccount(1024)

b = BankAccount(42)

result = a + b

print(result.balance)

* o/p: 1066
* \_\_truediv\_\_ defines SpecialString
* class SpecialString:

    def \_\_init\_\_(self, cont):

        self.cont = cont

    def \_\_truediv\_\_(self, other):

        line = "=" \* len(other.cont)

        return "\n".join([self.cont, line, other.cont])

spam = SpecialString("spam")

hello = SpecialString("Hello world!")

print(spam / hello)

* o/p:

spam

============

Hello world!

* Here, division operation is defined for SpecialString class
* Python have magic methods for comparisons
* \_\_lt\_\_ for <
* \_\_le\_\_ for <=
* \_\_eq\_\_ for ==
* \_\_ne\_\_ for !=
* \_\_gt\_\_ for >
* \_\_ge\_\_ for >=
* Here, if \_\_ne\_\_ isn’t implemented, it returns the opposite of \_\_eq\_\_
* class SpecialString:

    def \_\_init\_\_(self, cont):

        self.cont = cont

    def \_\_gt\_\_(self, other):

        for index in range(len(other.cont)+1):

            result = other.cont[:index] + ">" + self.cont

            result += ">" + other.cont[index:]

            print(result)

spam = SpecialString("spam")

eggs = SpecialString("eggs")

spam > eggs

* o/p:

>spam>eggs

e>spam>ggs

eg>spam>gs

egg>spam>s

eggs>spam>

* As you can see, any custom behaviour for the overloaded operators can be defined
* Several magic methods make classes act like containers
* \_\_len\_\_ for len()
* \_\_getitem\_\_ for indexing
* \_\_setitem\_\_ for assigning to indexed values
* Ex: x[y] = z is made by x.\_\_setitem\_\_(y, z)
* \_\_delitem\_\_ for deleting indexed values
* \_\_iter\_\_ for iteration over objects (Ex: in for loops)
* \_\_contains\_\_ for in
* \_\_call\_\_ for calling objects as functions
* \_\_int\_\_, \_\_str\_\_ and for converting objects to built in types
* import random

class VagueList:

    def \_\_init\_\_(self, cont):

        self.cont = cont

    def \_\_getitem\_\_(self, index):

        return self.cont[index + random.randint(-1, 1)]

    def \_\_len\_\_(self):

        return random.randint(0, len(self.cont)\*2)

vague\_list = VagueList(["A", "B", "C", "D", "E"])

print(len(vague\_list))

print(len(vague\_list))

print(vague\_list[2])

print(vague\_list[2])

* o/p:

10

9

C

C

* This outputs differs at every run: due to random number
* Here, len() is overridden to return a random number for class VagueList
* indexing function also returns a random item in a range from the list
* Operator overloading:

class Shape:

    def \_\_init\_\_(self, w, h):

        self.width = w

        self.height = h

    def area(self):

        return self.width\*self.height

    def \_\_add\_\_(self, other):

        return Shape(self.width + other.width, self.height + other.height)

    def \_\_gt\_\_(self, other):

        return self.area() > other.area()

w1 = int(input())

h1 = int(input())

w2 = int(input())

h2 = int(input())

s1 = Shape(w1, h1)

s2 = Shape(w2, h2)

result = s1 + s2

print(result.area())

print(s1 > s2)

Object lifecycle

* Lifecycle of creation: creation, manipulation, destruction
* First stage of lifecycle: definition of class which it belongs
* Next stage: Instantiation of an instance Ex: when \_\_init\_\_ is called
* Here, memory is allocated to store the instance
* Before that, \_\_new\_\_ method of class is called
* This is usually overridden only in special cases
* After this, object is ready to be used
* After finish being used, it can be destroyed
* Here, the memory that was allocated is freed up
* Destruction occurs when reference count reaches zero
* Reference count: number of variables and other elements that refer to an object
* If nothing is referring to it: Zero reference count
* Then, it can be safely deleted
* del statement reduces reference count of object one by once and leads to deletion
* Magic method for del: \_\_del\_\_
* Process of deleting objects when they are no longer needed: garbage collection
* Object’s reference count increases when it’s assigned a new name or placed in container (list, tuple, dictionary)
* Reference count decreases when it’s deleted with del
* When it reaches zero Python automatically deletes it
* Low level languages like C don’t have automatic memory management
* a = 42 # Create object <42>

b = a # Increase ref. count of <42>

c = [a] # Increase ref. count of <42>

del a # Decrease ref. count of <42>

b = 100 # Decrease ref. count of <42>

c[0] = -1 # Decrease ref. count of <42>

Encapsulation: Data hiding

* This is encapsulation: Implementation details of a class should be hidden
* Other language use private methods
* Python doesn’t have such method
* \_ is used at the begining to indicate it’s private
* from module\_name import won’t import variables starting with \_
* class Queue:

    def \_\_init\_\_(self, contents):

        self.\_hiddenlist = list(contents)

    def push(self, value):

        self.\_hiddenlist.insert(0, value)

    def pop(self):

        return self.\_hiddenlist.pop(-1)

    def \_\_repr\_\_(self):

        return "Queue({})".format(self.\_hiddenlist)

queue = Queue([1, 2, 3])

print(queue)

queue.push(0)

print(queue)

queue.pop()

print(queue)

print(queue.\_hiddenlist)

* o/p:

Queue([1, 2, 3])

Queue([0, 1, 2, 3])

Queue([0, 1, 2])

[0, 1, 2]

* Double underscores are used in the beginning to mark it as strongly private
* This is done to avoid bugs if there’re subclasses that have methods or attributes with the same names
* They can still be accessed externally
* Ex: \_\_privatemethod of Spam class can be accessed with \_Spam\_\_privatemethod
* class Player:

    def \_\_init\_\_(self, name, lives):

        self.name = name

        self.\_lives = lives

    def hit(self):

        self.\_lives -= 1

        if self.\_lives == 0:

            print("Game Over")

p = Player("Cyberpunk77", 3)

p.hit()

p.hit()

p.hit()

Class methods

* Class methods are called by a class, which is passed to cls parameter of the method
* Class methods are marked with a @classmethod decorator
* class Rectangle:

    def \_\_init\_\_(self, width, height):

        self.width = width

        self.height = height

    def calculate\_area(self):

        return self.width \* self.height

    @classmethod

    def new\_square(cls, side\_length):

        return cls(side\_length, side\_length)

square = Rectangle.new\_square(5)

print(square.calculate\_area())

* o/p: 25
* Here, new\_square is a class method
* It returns a new object of the class cls
* self and cls are just conventions
* class Person:

def \_\_init\_\_(self, name):

self.name = name

@classmethod

def sayHi(cls):

print(“Hi”)

Static methods

* Similar to class methods but don’t receive any additional arguments
* Identical to normal functions that belong to a class
* They are marked with @staticmethod decorator
* class Pizza:

    def \_\_init\_\_(self, toppings):

        self.toppings = toppings

    @staticmethod

    def validate\_topping(topping):

        if topping == "pineapple":

            raise ValueError("No pineapples!")

        else:

            return True

ingredients = ["cheese", "onions", "spam"]

if all(Pizza.validate\_topping(i) for i in ingredients):

    pizza = Pizza(ingredients)

* o/p: No output.
* Static methods behave like plain functions but you can call them from an instance of the class
* class Shape:

    def \_\_init\_\_(self, width, height):

        self.width = width

        self.height = height

    @staticmethod

    def area(weight, height):

        return weight\*height

w = int(input())

h = int(input())

print(Shape.area(w, h))

Properties

* Properties provide a way of customizing access to instance attributes
* Created by putting @property decorator above a method
* When the instance attribute with the same name as the method is accessed, the method will be called instead
* This makes an attribute read-only
* class Pizza:

    def \_\_init\_\_(self, toppings):

        self.toppings = toppings

    @property

    def pineapple\_allowed(self):

        return False

pizza = Pizza(["cheese", "tomato"])

print(pizza.pineapple\_allowed)

pizza.pineapple\_allowed = True

* class Person:

def \_\_init\_\_(self, age):

self.age = int(age)

@property

def isAdult(self):

if self.age > 18:

return True

else:

return False

* Properties can be set by defining getter/setter functions
* setter: Sets property’s value
* getter: Gets the value
* Define a setter: Use a decorator of the same name as the property followed by .setter
* Use the same method for getter
* class Pizza:

    def \_\_init\_\_(self, toppings):

        self.toppings = toppings

        self.\_pineapple\_allowed = False

    @property

    def pineapple\_allowed(self):

        return self.\_pineapple\_allowed

    @pineapple\_allowed.setter

    def pineapple\_allowed(self, value):

        if value:

            password = input("Enter the password: ")

            if password == "Sw0rdf1sh!":

                self.\_pineapple\_allowed = value

            else:

                raise ValueError("Alert! Intruder!")

pizza = Pizza(["cheese", "tomato"])

print(pizza.pineapple\_allowed)

pizza.pineapple\_allowed = True

print(pizza.pineapple\_allowed)

* class Player:

    def \_\_init\_\_(self, name, lives):

        self.name = name

        self.\_lives = lives

    def hit(self):

        self.\_lives -= 1

    @property

    def isAlive(self):

        if self.\_lives > 0:

            return True

p = Player("Cyberpunk77", int(input()))

i = 1

while True:

    p.hit()

    print("Hit # " + str(i))

    i += 1

    if not p.isAlive:

        print("Game Over")

        break

* class Person:

def \_\_init\_\_(self, name):

self.\_name = name

@property

def name(self):

return self.\_name

@name.setter

def name(self, value):

self.\_name = value

* Shooting game:

class Enemy:

  name = ""

  lives = 0

  def \_\_init\_\_(self, name, lives):

    self.name = name

    self.lives = lives

  def hit(self):

    self.lives -= 1

    if self.lives <= 0:

       print(self.name + ' killed')

    else:

        print(self.name + ' has '+ str(self.lives) + ' lives')

class Monster(Enemy):

  def \_\_init\_\_(self):

    super().\_\_init\_\_('Monster', 3)

  def hit(self):

    super().hit()

class Alien(Enemy):

  def \_\_init\_\_(self):

    super().\_\_init\_\_('Alien', 5)

  def hit(self):

    super().hit()

m = Monster()

a = Alien()

while True:

    x = input()

    if x == 'exit':

        break

    elif x== "laser":

      a.hit()

    elif x == "gun":

      m.hit()

Groups

* 2 useful groups: named groups and non-capturing groups
* Named groups format: (?P<name>...)
* Here, name is the name of the group and ... is the content
* Non capturing groups have the format: (?:...)
* import re

pattern = r"(?P<first>abc)(?:def)(ghi)"

match = re.match(pattern, "abcdefghi")

if match:

    print(match.group("first"))

    print(match.groups())

* o/p:

abc

('abc', 'ghi')

Metacharacters

* | means or
* Ex: red|blue matches either “red” or “blue”
* import re

pattern = r"gr(a|e)y"

match = re.match(pattern, "gray")

if match:

    print ("Match 1")

match = re.match(pattern, "grey")

if match:

    print ("Match 2")

match = re.match(pattern, "griy")

if match:

     print ("Match 3")

* o/p:

Match 1

Match 2

* This is [aeiou] similar to (a|e|i|o|u)

Special sequences

* Written as \ followed by another character
* Ex: \17 matches the experssion of the group of that number
* import re

pattern = r"(.+) \1"

match = re.match(pattern, "word word")

if match:

    print ("Match 1")

match = re.match(pattern, "?! ?!")

if match:

    print ("Match 2")

match = re.match(pattern, "abc cde")

if match:

    print ("Match 3")

* o/p:

Match 1

Match 2

* Here, \1 refers to first group’s subexpression
* Ex: (abc|xyz)\1 match “abc” or “xyz”, followed by the same thing
* Useful special sequences: \d for digits, \s for whitesapces and \w for word characters
* In ASCII mode, this is equivalent to [0-9], [\t\n\r\f\v], and [a-zA-Z0-9\_]
* Also: \D, \S and \W mean opposite of lower case versions
* Ex: \D matches non-digits
* import re

pattern = r"(\D+\d)"

match = re.match(pattern, "Hi 999!")

if match:

    print("Match 1")

match = re.match(pattern, "1, 23, 456!")

if match:

    print("Match 2")

match = re.match(pattern, " ! $?")

if match:

    print("Match 3")

* o/p: Match 1
* import re

text = input()

pattern = r"#\w+"

match = re.findall(pattern, text)

for i in match:

    print(i)

* (\d\*\W)+ and [1-6!] matches pattern “123!456!”?
* Additional special sequences: \A for beginning of string, \Z for end of string and \b for empty string between \w and \W characters
* HackerRank: Regex and parsing - Matrix Script

import math

import os

import random

import re

import sys

first\_multiple\_input = input().rstrip().split()

n = int(first\_multiple\_input[0])

m = int(first\_multiple\_input[1])

matrix = []

for \_ in range(n):

    matrix\_item = input()

    matrix.append(matrix\_item)

pattern = r"(?<=\w)[!@#$%& ]{1,}(?=\s\*\w)"

firstletter = ""

for col in range(m):

    for row in range(n):

        firstletter += str(matrix[row][col])

print(re.sub(pattern, " ", firstletter))

Python data structures

* Built-in data structures: Strings, lists, dictionaries, tuples, sets, etc
* Most prominent data structures: Stacks, queues, trees, linked lists
* Below code counts number of vowels of input text
* inputText = input()

total = 0

for i in inputText:

    if i == "a" or i == "e" or i == "i" or i == "o" or i == "u":

        total += 1

print(total)

HackerRank: Find the runner-up score!

* n = int(input())

arr = list(set(map(int, input().split())))

maxy = max(arr)

arr.remove(maxy)

run = max(arr)

print(run)

HackerRank - Finding the percentage

* n = int(input())

     student\_marks = {}

     for \_ in range(n):

        name, \*line = input().split()

        scores = list(map(float, line))

        student\_marks[name] = scores

     query\_name = input()

     total = student\_marks[query\_name]

     avg = 0

     for val in total:

        avg += val

     avg = avg/3

     print("{:.2f}".format(avg))

HackerRank - Designer door mat

* nm = input().split()

n = int(nm[0])

m = int(nm[1])

for i in range(n//2):

    print((".|."\*(2\*i+1)).center(m, "-"))

print("WELCOME".center(m, "-"))

arr = []

for i in range(n//2):

    arr.append(str(".|."\*(2\*i+1)).center(m, "-"))

arr.reverse()

for i in arr:

    print(i)

HackerRank: XML2 - Find the maximum depth

* import xml.etree.ElementTree as etree

maxdepth = 0

def depth(elem, level):

    global maxdepth

    if (level == maxdepth):

        maxdepth += 1

    for child in elem:

        depth(child, level + 1)

if \_\_name\_\_ == '\_\_main\_\_':

    n = int(input())

    xml = ""

    for i in range(n):

        xml =  xml + input() + "\n"

    tree = etree.ElementTree(etree.fromstring(xml))

    depth(tree.getroot(), -1)

    print(maxdepth)

HackerRank: sWAP cASE

* def swap\_case(s):

    final = ""

    for i in s:

        upper = i.upper()

        lower = i.lower()

        if i == upper:

            final += i.lower()

        elif i == lower:

            final += i.upper()

        else:

            final += i

    return final

if \_\_name\_\_ == '\_\_main\_\_':

    s = input()

    result = swap\_case(s)

    print(result)

HackerRank: String split and join

def split\_and\_join(line):

*# write your code here*

    arr = line.split()

    a = "-".join(arr)

    return a

if \_\_name\_\_ == '\_\_main\_\_':

    line = input()

    result = split\_and\_join(line)

    print(result)

HackerRank: The Minion Game

*#!/bin/python3*

**import** math

**import** os

**import** random

**import** re

**import** sys

**def** solve(s):

**return** " ".join([i.capitalize() **for** i **in** s.split(" ")])

**if** \_\_name\_\_ == '\_\_main\_\_':

    fptr = **open**(os.environ['OUTPUT\_PATH'], 'w')

    s = **input**()

    result = solve(s)

    fptr.write(result + '\n')

    fptr.close()

HackerRank: Hourglass Sum

* *#!/bin/python3*

**import** math

**import** os

**import** random

**import** re

**import** sys

**def** hourglassSum(arr):

    max\_sum = []

**for** i **in** **range**(0, 4):

**for** j **in** **range**(0, 4):

            sumHg = (arr[i][j] + arr[i][j + 1] + arr[i][j + 2]) + (arr[i + 1][j + 1]) + (arr[i + 2][j] + arr[i + 2][j + 1] + arr[i + 2][j + 2])

            max\_sum.append(sumHg)

**return** **max**(max\_sum)

**if** \_\_name\_\_ == '\_\_main\_\_':

    fptr = **open**(os.environ['OUTPUT\_PATH'], 'w')

    arr = []

**for** \_ **in** **range**(6):

        arr.append(**list**(**map**(**int**, **input**().rstrip().split())))

    result = hourglassSum(arr)

    fptr.write(**str**(result) + '\n')

    fptr.close()