In [10]:
```
x=2
y=-3
x=x+y
y=x-y
x=x-y
print(x,y)
```

```
-3 2
```

Palindrome

In [21]:
```
a = input("Enter name  : ")
b= a[::-1] #reversed string
print(b)
if b == a:
    print("True")
else:
    print("false")
```

```
Enter name  : gaurav
varuag
false
```

f print functionality

In [22]:
```
num = int(input("number : "))
for i in range(1,11):
    print(f"{num}x{i} = {1*num}")  # (f) allows writing variables between strings
```

```
number : 5
5x1 = 5
5x2 = 5
5x3 = 5
5x4 = 5
5x5 = 5
5x6 = 5
5x7 = 5
5x8 = 5
5x9 = 5
5x10 = 5
```

In [25]:
```
for i in range(4):
    print("*"*(i+1))
```

```
*
**
***
****
```

use of .format()

In [37]:
```
capitals  = {'France' : 'Paris',
            'China' : 'Brijing',
            'Italy' : 'Rome',
            'Russia': 'Moscow'}
for country in capitals:  # the "in" keyword brings all the keys, never values
    print("{},{}".format(capitals[country], country))

if 'China' in capitals.keys():
    print(True)
if 'Rome' in capitals.values():
    print(True)
```

```
Paris,France
Brijing,China
Rome,Italy
Moscow,Russia
True
True
```

In [3]:

```
"""
Tabular data as a nested list.
"""
# Programming language popularity, from www.tiobe.com/tiobe-index
popularity = [["Language", 2017, 2012, 2007, 2002, 1997, 1992, 1987],
              ["Java", 1, 2, 1, 1, 15, 0, 0],
              ["C", 2, 1, 2, 2, 1, 1, 1],
              ["C++", 3, 3, 3, 3, 2, 2, 5],
              ["C#", 4, 4, 7, 13, 0, 0, 0],
              ["Python", 5, 7, 6, 11, 27, 0, 0],
              ["Visual Basic .NET", 6, 17, 0, 0, 0, 0, 0],
              ["PHP", 7, 6, 4, 5, 0, 0, 0],
              ["JavaScript", 8, 9, 8, 7, 23, 0, 0],
              ["Perl", 9, 8, 5, 4, 4, 10, 0]]

# < = left aligns the values
# > = right aligns the values
# ^ = center aligns the values

format_string = "{:<20}  {:>4}  {:>4}  {:>4}  {:>4}  {:>4}  {:>4}  {:>4}"

# Display langauges table
headers = popularity[0]

header_row = format_string.format(*headers)
# since headers is a list, the * turns the list into 8(here) individual arguments for .format

print(header_row)
print("-" * len(header_row))

for language in popularity[1:]:
    print(format_string.format(*language))

print("")

# Finding/selecting items

# What was Python's popularity in 1997?
print("Python's popularity in 1997:", popularity[5][5])

def find_col(table, col):
    """
    Return column index with col header in table
    or -1 if col is not in table
    """
    return table[0].index(col)

def find_row(table, row):
    """
    Return row index with row header in table
    or -1 if row is not in table
    """
    for idx in range(len(table)):
        if table[idx][0] == row:
            return idx
    return -1

idx1997 = find_col(popularity, 1997)
idxpython = find_row(popularity, "Python")
print("Python's popularity in 1997:", popularity[idxpython][idx1997])
```

```
Language              2017  2012  2007  2002  1997  1992  1987
------------------------------------------------------------
Product               Vendor        Type               Vulnerabilities
-----------------------------------------------------------------------
Android               Google        Operating System        564
Linux Kernel          Linux         Operating System        367
Imagemagick           Imagemagick   Application             307
IPhone OS             Apple         Operating System        290
Mac OS X              Apple         Operating System        210
Windows 10            Microsoft     Operating System        195
Windows Server 2008   Microsoft     Operating System        187
Windows Server 2016   Microsoft     Operating System        183
Windows Server 2012   Microsoft     Operating System        176
Windows 7             Microsoft     Operating System        174

174
Android 564
Perl                      9     8     5     4     4    10     0

Python's popularity in 1997: 27
Python's popularity in 1997: 27
```

In [1]:
```
a={}  #this creates empty dictionary
# this does not create empty SET

b = set()   # this creates  an empty set
b.add(4)
b.add(7)   # we cannot add lists or dictionary to sets,   only TUPLES allowed
```

In [3]:
```
mydict = {"21":"gahg", "gaur":"lion", 'ran':"loop","555":"tups",453:"21",1:" ** ", 2:{'a','b',
print(mydict['gaur'])
print(mydict['555'])
print(mydict['ran'])
print(mydict[453]) #numbers can e written without quotes, inside the dictionary as well as whi
#inside a dictionary, a string cannot be written without qoutes or double quotes.
#there is no such thing as index of dictionary like 0,1,2,3,4,5(UNORDERED). They are indexed c

print(type(mydict.keys())) #the return type is "dict_keys"
print(mydict.keys())       # this returns a TUPLE
print(mydict.values())
print(list(mydict.keys())) # can be converted to LIST

#print(mydict.items())

mydict.update({'abc':'xyz'})  #adds new key-value pair at the end.
print(mydict['abc'])

print(mydict.get('harry2'))  #this method returns NONE when the key is not found
#print(mydict['harry2'])      #this statement throws an ERROR, as shown in the output

# so try to use .get() function
```

```
lion
tups
loop
21
<class 'dict_keys'>
dict_keys(['21', 'gaur', 'ran', '555', 453, 1, 2])
dict_values(['gahg', 'lion', 'loop', 'tups', '21', ' ** ', {'a', 'c', 'b'}])
['21', 'gaur', 'ran', '555', 453, 1, 2]
xyz
None
```

In [4]:
```
"""
Tabular data as nested dictionaries.
"""
# Top 10 software products with the most vulnerabilities in 2017
# (through August).  From www.cvedetails.com.
vulnerabilities2017 = {
    'Android': {'vendor': 'Google',
                'type': 'Operating System',
```

```
                'number': 564},
    'Linux Kernel': {'vendor': 'Linux',
                     'type': 'Operating System',
                     'number': 367},
    'Imagemagick': {'vendor': 'Imagemagick',
                    'type': 'Application',
                    'number': 307},
    'IPhone OS': {'vendor': 'Apple',
                  'type': 'Operating System',
                  'number': 290},
    'Mac OS X': {'vendor': 'Apple',
                 'type': 'Operating System',
                 'number': 210},
    'Windows 10': {'vendor': 'Microsoft',
                   'type': 'Operating System',
                   'number': 195},
    'Windows Server 2008': {'vendor': 'Microsoft',
                            'type': 'Operating System',
                            'number': 187},
    'Windows Server 2016': {'vendor': 'Microsoft',
                            'type': 'Operating System',
                            'number': 183},
    'Windows Server 2012': {'vendor': 'Microsoft',
                            'type': 'Operating System',
                            'number': 176},
    'Windows 7': {'vendor': 'Microsoft',
                  'type': 'Operating System',
                  'number': 174}
}

# Display vulnerabilities table
print("Product               Vendor        Type               Vulnerabilities")
print("-------------------------------------------------------------------------")

for product, values in vulnerabilities2017.items():
    row = "{:21} {:13} {:18} {:8}".format(product, values['vendor'], values['type'], values['r
    print(row)

print("")

# Finding/selecting items

# How many vulnerabilites does Windows 7 have?
print(vulnerabilities2017['Windows 7']['number'])

# What product had the most vulnerabilities?
maxproduct = None
maxnumber = -1

for product, values in vulnerabilities2017.items():
    if values['number'] > maxnumber:
        maxproduct = product
        maxnumber = values['number']

print(maxproduct, maxnumber)
```

```
Product               Vendor        Type               Vulnerabilities
-------------------------------------------------------------------------
Android               Google        Operating System        564
Linux Kernel          Linux         Operating System        367
Imagemagick           Imagemagick   Application             307
IPhone OS             Apple         Operating System        290
Mac OS X              Apple         Operating System        210
Windows 10            Microsoft     Operating System        195
Windows Server 2008   Microsoft     Operating System        187
Windows Server 2016   Microsoft     Operating System        183
Windows Server 2012   Microsoft     Operating System        176
Windows 7             Microsoft     Operating System        174

174
Android 564
```

dictionary wrappers and iterators

In [1]:
```
"""
Example code for printing the contents of a dictionary to the console
"""
NAME_DICT = {"Warren" : "Joe", "Rixner" : "Scott", "Greiner" : "John"}

def run_dict_methods():
    """
    Run some simple examples of calls to dictionary methods
    """

    # Note that these methods return an iterable object (similar to range())
    print(NAME_DICT.keys())
    print(NAME_DICT.values())
    print(NAME_DICT.items())
    print()

    # These objects can be converted to lists
    print(list(NAME_DICT.keys()))
    print(list(NAME_DICT.values()))
    print(list(NAME_DICT.items()))

run_dict_methods()

def print_dict_keys(my_dict):
    """
    Print the contents of a dictionary to the console
    in a readable form using the keys() method
    """
    print("Printing dictionary", my_dict, "in readable form")
    for key in my_dict:                              # note my_dict.keys() works here too
        print("Key =", key, "has value =", my_dict[key])


def print_dict_items(my_dict):
    """
    Print the contents of a dictionary to the console
    in a readable form using the items() method
    """
    print("Printing dictionary", my_dict, "in readable form")
    for (key, value) in my_dict.items():
        print("Key =", key, "has value =", value)


def run_print_dict_examples():
    """
    Run some examples of printing dictionaries to the console
    """
    print()
    print_dict_keys(NAME_DICT)
    print()
    print_dict_items(NAME_DICT)

#run_print_dict_examples()
"""
The dict_keys, dict_values, dict_items are the WRAPPERS of keys, values and items.
These full statements are the ITERATORS returned by python3.
These wrappers can be stripped off and the content can be returned as a list,
as seen in the next few lines of output.
"""
```

```
dict_keys(['Warren', 'Rixner', 'Greiner'])
dict_values(['Joe', 'Scott', 'John'])
dict_items([('Warren', 'Joe'), ('Rixner', 'Scott'), ('Greiner', 'John')])

['Warren', 'Rixner', 'Greiner']
['Joe', 'Scott', 'John']
[('Warren', 'Joe'), ('Rixner', 'Scott'), ('Greiner', 'John')]
```

Nested List

In [1]:
```
"""
Solution - Create a list zero_list consisting of 3 zeroes using a list comprehension
https://docs.python.org/3/tutorial/datastructures.html#list-comprehensions

As a challenge, redo the previous problem using a nested list comprehension
https://docs.python.org/3/tutorial/datastructures.html#nested-list-comprehensions
"""

# Add code here for a list comprehension
zero_list = [0 for dummy_idx in range(3)]

# Add code here for nested list comprehension
nested_list = [[0 for dummy_idx1 in range(3)] for dummy_idx2 in range(5)]


# Tests
print(zero_list)
print(nested_list)

# Output
#[0, 0, 0]
#[[0, 0, 0], [0, 0, 0], [0, 0, 0], [0, 0, 0], [0, 0, 0]]
```

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

In [2]:
```
"""
Solution - Select a specific item in a nested list
"""
# Define a nested list of lists
nested_list = [[col + 3 * row for col in range(3)] for row in range(5)]
print(nested_list)

# Add code to print out the item in this nested list with value 7
print(nested_list[2][1])

# Output
#[[0, 1, 2], [3, 4, 5], [6, 7, 8], [9, 10, 11], [12, 13, 14]]
#7
```

```
[[0, 1, 2], [3, 4, 5], [6, 7, 8], [9, 10, 11], [12, 13, 14]]
7
```

In [5]:
```
"""
Solution - Analyze a reference issue involving a nested list
"""
# Create a nested list
zero_list = [0, 2, 0]
nested_list = []
for dummy_idx in range(5):
    # nested_list.append(zero_list)
    nested_list.append(list(zero_list))    # Corrected code # zero_list is copied 5 times
print(nested_list)

# Update an entry to be non-zero
nested_list[2][1] = 7
print(nested_list)


# Erroneous output
#[[0, 0, 0], [0, 0, 0], [0, 0, 0], [0, 0, 0], [0, 0, 0]]
#[[0, 7, 0], [0, 7, 0], [0, 7, 0], [0, 7, 0], [0, 7, 0]]

# Desired output
# [[0, 2, 0], [0, 2, 0], [0, 2, 0], [0, 2, 0], [0, 2, 0]]
# [[0, 2, 0], [0, 2, 0], [0, 7, 0], [0, 2, 0], [0, 2, 0]]
```

```
# Explanation

# Line 13 is unintentionally updating all 5 entries in nested_list due to a referencing issue.

#Line 9 is creating five references to the SAME onject (the list zero_list) in nested_list.
#Thus, updating one reference to zero_list in nested_list in line 13 updates
#the other four references to zero_list in nested_list simultaneously.

# To visualize this reference issue in Python Tutor, visit the URL https://goo.gl/hT4MM3.
# Note the entries in nested_list all refer to SAME list.

# The solution to this problem is to make a NEW copy of zero_list each  time append()
# is executed. To do this, simply replace zero_list by list(zero_list) in line 9

# To visualize corrected code in Python Tutor, visit the URL https://goo.gl/4nifEg.
# Note that each entry in nested_list now refers to a DISTINCT list.  As a result,
# updates to one item in nested_list do not affect any other part of nested_list.
```

```
[[0, 2, 0], [0, 2, 0], [0, 2, 0], [0, 2, 0], [0, 2, 0]]
[[0, 2, 0], [0, 2, 0], [0, 7, 0], [0, 2, 0], [0, 2, 0]]
```

In [6]:

```
"""
Solution - Write a function dict_copies(my_dict, num_copies) that
returns a list consisting of num_copies copies of my_dict
"""
# Add code here
def dict_copies(my_dict, num_copies):
    """
    Given a dictionary my_dict and an integer num_copies,
    returns a list consisting of num_copies copies of my_dict.
    """
    answer = []
    for idx in range(num_copies):
        answer.append(dict(my_dict))
    return answer

# Tests
print(dict_copies({}, 0))
print(dict_copies({}, 1))
print(dict_copies({}, 2))

test_dict = dict_copies({'a' : 1, 'b' : 2}, 2)
print(test_dict)

# Check for reference problem
test_dict[1]["a"] = 3
print(test_dict)

# Output
#[]
#[{}]
#[{}, {}]
#[{'a': 1, 'b': 2}, {'b': 2, 'a': 1}]
#[{'b': 2, 'a': 1}, {'b': 2, 'a': 3}]

# Note that you have a reference issue if the last line of output is
#[{'a': 3, 'b': 2}, {'b': 2, 'a': 3}]
```

```
[]
[{}]
[{}, {}]
[{'a': 1, 'b': 2}, {'a': 1, 'b': 2}]
[{'a': 1, 'b': 2}, {'a': 3, 'b': 2}]
```

In [ ]:

```
"""
Solution - Create a function make_grade_table(names, grades_list)
whose keys are the names in names and whose values are the
lists of grades in grades
"""
# Add code here

def make_grade_table(name_list, grades_list):
```

```
    """
    Given a list of name_list (as strings) and a list of grades
    for each name, return a dictionary whose keys are
    the names and whose associated values are the lists of grades
    """

    grade_table = {}
    for name, grades in zip(name_list, grades_list):
        grade_table[name] = grades
    return grade_table

# Tests
print(make_grade_table([], []))

name_list = ["Joe", "Scott", "John"]
grades_list = [100, 98, 100, 13], [75, 59, 89, 77],[86, 84, 91, 78]
print(make_grade_table(name_list, grades_list))

# Output
#{}
#{'Scott': [75, 59, 89, 77], 'John': [86, 84, 91, 78], 'Joe': [100, 98, 100, 13]}
```

CSV as List

In [2]:

```
"""
Using the csv module.
"""
import csv

def parse(csvfilename):
    """
    Reads CSV file named csvfilename, parses
    it's content and returns the data within
    the file as a list of lists.
    """
    table = []
    with open(csvfilename, "r") as csvfile:
        csvreader = csv.reader(csvfile,
                               skipinitialspace=True)
        for row in csvreader:
            table.append(row)
    return table

def print_table(table):
    """
    Print out table, which must be a list
    of lists, in a nicely formatted way.
    """
    for row in table:
        # Header column left justified
        print("{:<19}".format(row[0]), end='')
        # Remaining columns right justified
        for col in row[1:]:
            print("{:>4}".format(col), end='')
        print("", end='\n')

table = parse("hightemp.csv")
print_table(table)

print("")
print("")

table2 = parse("hightemp2.csv")
print_table(table2)
```

```
City                Jan Feb Mar Apr May Jun Jul Aug Sep Oct Nov Dec
Houston              62  65  72  78  84  90  92  93  88  81  71  63
Baghdad              61  66  75  86  97 108 111 111 104  91  75  64
Moscow               21  25  36  50  64  72  73  70  59  46  34  25
San Francisco        57  60  62  63  64  66  67  68  70  69  63  57
London               43  45  50  55  63  68  72  70  66  57  50  45
Chicago              32  36  46  59  70  81  84  82  75  63  48  36
Sydney               79  79  77  73  68  64  63  64  68  72  75  79
Paris                45  46  54  61  68  73  77  77  70  61  52  46
Tokyo                46  48  54  63  70  75  82  84  79  68  59  52
Shanghai             46  48  55  66  75  81  90  90  81  72  63  52
```

```
City                Jan Feb Mar Apr May Jun Jul Aug Sep Oct Nov Dec
Houston, USA         62  65  72  78  84  90  92  93  88  81  71  63
Baghdad, Iraq        61  66  75  86  97 108 111 111 104  91  75  64
Moscow, Russia       21  25  36  50  64  72  73  70  59  46  34  25
San Francisco, USA   57  60  62  63  64  66  67  68  70  69  63  57
London, England      43  45  50  55  63  68  72  70  66  57  50  45
Chicago, USA         32  36  46  59  70  81  84  82  75  63  48  36
Sydney, Australia    79  79  77  73  68  64  63  64  68  72  75  79
Paris, France        45  46  54  61  68  73  77  77  70  61  52  46
Tokyo, Japan         46  48  54  63  70  75  82  84  79  68  59  52
Shanghai, China      46  48  55  66  75  81  90  90  81  72  63  52
```

CSV as Dictionary

In [3]:

```
"""
Using csv.DictReader.
"""
import csv

MONTHS = ('Jan', 'Feb', 'Mar', 'Apr',
          'May', 'Jun', 'Jul', 'Aug',
          'Sep', 'Oct', 'Nov', 'Dec')

def dictparse(csvfilename, keyfield):
    """
    Reads CSV file named csvfilename, parses
    it's content and returns the data within
    the file as a dictionary of dictionaries.
    """
    table = {}
    with open(csvfilename, "rt", newline='') as csvfile:
        csvreader = csv.DictReader(csvfile, skipinitialspace=True)
        for row in csvreader:
            table[row[keyfield]] = row
    return table

def print_table(table):
    """
    Print out table, which must be a dictionary
    of dictionaries, in a nicely formatted way.
    """
    print("City                ", end='')
    for month in MONTHS:
        print("{:>6}".format(month), end='')
    print("")
    for name, row in table.items():
        # Header column left justified
        print("{:<19}".format(name), end='')
        # Remaining columns right justified
        for month in MONTHS:
            print("{:>6}".format(row[month]), end='')
        print("", end='\n')

table = dictparse("hightemp.csv", 'City')
print_table(table)
```

City	Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep	Oct	Nov	Dec
Houston	62	65	72	78	84	90	92	93	88	81	71	63
Baghdad	61	66	75	86	97	108	111	111	104	91	75	64
Moscow	21	25	36	50	64	72	73	70	59	46	34	25
San Francisco	57	60	62	63	64	66	67	68	70	69	63	57
London	43	45	50	55	63	68	72	70	66	57	50	45
Chicago	32	36	46	59	70	81	84	82	75	63	48	36
Sydney	79	79	77	73	68	64	63	64	68	72	75	79
Paris	45	46	54	61	68	73	77	77	70	61	52	46
Tokyo	46	48	54	63	70	75	82	84	79	68	59	52
Shanghai	46	48	55	66	75	81	90	90	81	72	63	52

cipher Dictionary

In [ ]:

```
"""
Solution for parts 1-3
Using substitution ciphers to encrypt and decrypt plain text
"""
# Part 1 - Use a dictionary that represents a substition cipher to
# encrypt a phrase

# Example of a cipher dictionary 26 lower case letters plus the blank
CIPHER_DICT = {'e': 'u', 'b': 's', 'k': 'x', 'u': 'q', 'y': 'c', 'm': 'w', 'o': 'y',
               'g': 'f', 'a': 'm', 'x': 'j', 'l': 'n', 's': 'o', 'r': 'g', 'i': 'i',
               'j': 'z', 'c': 'k', 'f': 'p', ' ': 'b', 'q': 'r', 'z': 'e', 'p': 'v',
               'v': 'l', 'h': 'h', 'd': 'd', 'n': 'a', 't': ' ', 'w': 't'}

def encrypt(phrase, cipher_dict):
    """
    Take a string phrase (lower case plus blank)
    and encypt it using the dictionary cipher_dict
    """
    answer = ""
    for letter in phrase:
        answer += cipher_dict[letter]
    return answer

# Tests
print("Output for part 1")
print(encrypt("pig", CIPHER_DICT))
print(encrypt("hello world", CIPHER_DICT))
print()

# Output for part 1
#vif
#hunnybtygnd


# Part 2 - Compute an inverse substitution cipher that decrypts
# an encrypted phrase

def make_decipher_dict(cipher_dict):
    """
    Take a cipher dictionary and return the cipher
    dictionary that undoes the cipher
    """
    decipher_dict = {}
    for letter in cipher_dict:
        decipher_dict[cipher_dict[letter]] = letter
    return decipher_dict

DECIPHER_DICT = make_decipher_dict(CIPHER_DICT)

# Tests - note that applying encrypting with the cipher and decipher dicts
# should return the original results
print("Output for part 2")
print(DECIPHER_DICT)
print(encrypt(encrypt("pig", CIPHER_DICT), DECIPHER_DICT))
print(encrypt(encrypt("hello world", CIPHER_DICT), DECIPHER_DICT))
```

```
print()

# Output for part 2 - note order of items in dictionary is not important
"""
{'p': 'f', 'n': 'l', 'm': 'a', 'i': 'i', 'd': 'd', 'x': 'k',
'b': ' ', 'l': 'v', 'f': 'g', 'o': 's', 'u': 'e', 'a': 'n',
'c': 'y', 'r': 'q', 'e': 'z', 'k': 'c', 'w': 'm', 'g': 'r',
'y': 'o', ' ': 't', 'h': 'h', 'v': 'p', 'j': 'x', 'q': 'u', 't': 'w', 's': 'b', 'z': 'j'}
"""
#pig
#hello world

# Part 3 - Create a random cipher dictionary

import random

def make_cipher_dict(alphabet):
    """
    Given a string of unique characters, compute a random
    cipher dictionary for these characters
    """
    letters= list(alphabet)
    shuffled_letters = list(alphabet)
    random.shuffle(shuffled_letters)

    cipher_dict = {}
    for idx in range(len(alphabet)):
        letter = letters[idx]
        shuffled_letter = shuffled_letters[idx]
        cipher_dict[letter] = shuffled_letter
    return cipher_dict

# Tests
print("Output for part 3")
print(make_cipher_dict(""))
print(make_cipher_dict("cat"))
print(make_cipher_dict("abcdefghijklmnopqrstuvwxyz "))

# Output for part 3 -  note that answers are randomized
#{}
#{'a': 'a', 't': 'c', 'c': 't'}
"""
{'a': 'h', 'l': 'u', 'u': 'q', 'b': 'v', 'y': 'a', 'm': 'r',
'p': 'j', 'k': 'e', 'n': 'p', 't': 'x', 'd': 'o', 'c': 'c', 'w': ' ',
'f': 'd', 'r': 'z', 'v': 'l', 's': 'y', 'e': 'b', 'o': 'i', 'x': 'm',
'h': 's', 'i': 'w', 'q': 'g', 'g': 'n', 'j': 'f', 'z': 'k', ' ': 't'}
```