Table of contents

1. if - if statements

(go to top)

In [4]:
```
# convert2.py
# A program to convert Celsius temps to Fahrenheit.
# This version issues heat and cold warnings.

def main():
    celsius = float(input("What is the Celsius temperature? "))
    fahrenheit = 9/5 * celsius + 32
    print("The temperature is", fahrenheit, "degrees Fahrenheit.")

    # ---------------------------------------------

    # Print warnings for extreme temps
    if fahrenheit > 90:
        print("It's really hot out there. Be careful!")
    if fahrenheit < 30:
        print("Brrrrr. Be sure to dress warmly!")

main()
```

```
What is the Celsius temperature? 50
The temperature is 122.0 degrees Fahrenheit.
It's really hot out there. Be careful!
```

2. if - else statements

(go to top)

In [5]:
```
room = 'bed'
area = 14.0
```

In [6]:
```
if room == 'kit':
    print('looking around in the kitchen')
else:
    print('looking around elsewhere')
```

```
looking around elsewhere
```

In [ ]:
```

```

3. if - elif - else statements

(go to top)

In [4]:
```
room = 'bed'
area = 14.0
```

In [5]:
```
if room == 'kit':
    print('looking around in the kitchen')
elif room == 'bed':
    print('looking around in the bedroom')
else:
    print('looking around elsewhere')
```

```
looking around in the bedroom
```

In [7]:
```
if area > 15:
    print('big place!')
elif area > 10:
    print('medium size, nice!')
else:
    print('pretty small')
```

```
medium size, nice!
```

In [16]:
```
import math
def main():
    print("This program finds the real solutions to a quadratic\n")

    a = float(input("Enter coefficient a: "))
    b = float(input("Enter coefficient b: "))
    c = float(input("Enter coefficient c: "))

    discrim = b * b - (4 * a * c)
    if discrim < 0:
        print("\nThe equation has no real roots!")
    elif discrim == 0:
        root = -b / (2 * a)
        print("\nThere is a double root at", root)
    else:
        discRoot = math.sqrt(b * b - 4 * a * c)
        root1 = (-b + discRoot) / (2 * a)
        root2 = (-b - discRoot) / (2 * a)
        print("\nThe solutions are:", '{0:0.2f} , {1:0.2f}'.format(roo

main()
```

```
This program finds the real solutions to a quadratic

Enter coefficient a: 3
Enter coefficient b: 7
Enter coefficient c: 2

The solutions are: -0.33 , -2.00
```

---

4. Exception / Error handling

- A try statement has the general form:

```
try:
    <body>
except <ErrorType>:
    <handler>
```

(go to top)

In [24]:
```
def sqrt(x):
    """Returns the sq root of a number"""
    try:
        return x ** 0.5
    except:
        print('x must be a float or int')

print(sqrt('hello'))
```

```
x must be a float or int
None
```

In [26]:
```
print(sqrt(4))
```

```
2.0
```

4.1. Catch Type Errors only

- When an operation or function is applied to an object of innapropriate type etc.....

(go to top)

In [28]:
```
def sqrt(x):
    """Returns the sq root of a number"""
    try:
        return x ** 0.5
    except TypeError:
        print('x must be a float or int')

print(sqrt('hello'))
```

```
x must be a float or int
None
```

In [22]:
```
print(sqrt(4))
```

```
2.0
```

4.2. Raise an Error

- Sometimes an input might run perfectly in python
- But if we dont want to allow that kind of input we use raise

(go to top)

In [25]:
```
def sqrt(x):
    """Returns the sq root of a number"""
    if type(x) != type(2):
        raise TypeError('x must be a float of int')
    elif x < 0:
        raise ValueError('x must be non-negative')
    return x ** 0.5
```

In [28]:
```
print(sqrt('x'))
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-28-9344b316444c> in <module>
----> 1 print(sqrt('x'))

<ipython-input-25-d836447f87e7> in sqrt(x)
      2     """Returns the sq root of a number"""
      3     if type(x) != type(2):
----> 4         raise TypeError('x must be a float of int')
      5     elif x < 0:
      6         raise ValueError('x must be non-negative')

TypeError: x must be a float of int
```

In [30]:
```
print(sqrt(-2))
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-30-4f4356609c54> in <module>
----> 1 print(sqrt(-2))

<ipython-input-25-d836447f87e7> in sqrt(x)
      4         raise TypeError('x must be a float of int')
      5     elif x < 0:
----> 6         raise ValueError('x must be non-negative')
      7     return x ** 0.5

ValueError: x must be non-negative
```

4.2. Example

- The multiple excepts are similar to elifs. If an error occurs, Python will try each except in tum looking for one that matches the type of error. The bare except at the bottom in this example acts like an else and will be used as the default if no previous except error type matches. If there is no default at the bottom and none of the except types match the error, then the program crashes and Python reports the error.
- If you follow the error type with an `as <variable>` in an except clause, Python will assign that variable the actual exception object. In this case, I turned the exception into a string and looked at the message to see what caused the ValueError. Notice that this text is exactly what Python prints out if the error is not caught (e.g., `ValueError: math domain error`).

(go to top)

In [47]:
```
def main():
    print("This program finds the real solutions to a quadratic\n")

    a = float(input("Enter coefficient a: "))
    b = float(input("Enter coefficient b: "))
    c = float(input("Enter coefficient c: "))
    discRoot = math.sqrt(b * b - 4 * a * c)
    root1 = (-b + discRoot) / (2 * a)
    root2 = (-b - discRoot) / (2 * a)
    print("\nThe solutions are:", root1, root2 )

main()
#enter 2,2,6 for a,b,c
# note ValueError: math domain error
```

```
Enter coefficient b: 2
Enter coefficient c: 6

---------------------------------------------------------------------------
--------
ValueError                                Traceback (most recent ca
ll last)
<ipython-input-47-392f6f16faa2> in <module>
     10     print("\nThe solutions are:", root1, root2 )
     11
---> 12 main()
     13 #enter 2,2,6 for a,b,c

<ipython-input-47-392f6f16faa2> in main()
      5     b = float(input("Enter coefficient b: "))
      6     c = float(input("Enter coefficient c: "))
----> 7     discRoot = math.sqrt(b * b - 4 * a * c)
      8     root1 = (-b + discRoot) / (2 * a)
      9     root2 = (-b - discRoot) / (2 * a)

ValueError: math domain error
```

In [48]:
```
def main():
    try:
        print("This program finds the real solutions to a quadratic\n"

        a = float(input("Enter coefficient a: "))
        b = float(input("Enter coefficient b: "))
        c = float(input("Enter coefficient c: "))
        discRoot = math.sqrt(b * b - 4 * a * c)
        root1 = (-b + discRoot) / (2 * a)
        root2 = (-b - discRoot) / (2 * a)
        print("\nThe solutions are:", root1, root2 )
    except ValueError as v_err:
        if str(v_err) =="math domain error":
            print("No Real Roots")
        else:
            print("Invalid coefficient given")
    except :
        print("\nSomething went wrong, sorry!")

main()
#enter 2,2,6 for a,b,c
```

```
This program finds the real solutions to a quadratic

Enter coefficient a: 2
Enter coefficient b: k
Invalid coefficient given
```

---

5. For Loops

(go to top)

5.1. Strings as Range

(go to top)

In [2]:
```
message = 'Man'
```

In [3]:
```
for i in message:
    print(i)
```

```
M
a
n
```

5.2. Range as Range

(go to top)

In [4]:
```
for i in range(3):
    print(i)
```

```
0
1
2
```

- range 0 to 10, step of 2

In [5]:
```
for i in range(0,10,2):
    print(i)
```

```
0
2
4
6
8
```

5.3. Nested List as Range

(go to top)

In [6]:
```
house = [['hallway', 11.25],['kitchen', 18.0], ['kiving room', 20.0],
```

In [7]:
```
for i in house:
    print(i)
```

```
['hallway', 11.25]
['kitchen', 18.0]
['kiving room', 20.0]
['bedroom', 10.75]
['bathroom', 9.5]
```

- for the first iteration i is ['hallway', 11.25]. therefor i[0] is "hallway"

In [8]:
```
for i in house:
    print('the ' + i[0] + ' is ' + str(i[1]) + ' sqm')
```

```
the hallway is 11.25 sqm
the kitchen is 18.0 sqm
the kiving room is 20.0 sqm
the bedroom is 10.75 sqm
the bathroom is 9.5 sqm
```

5.4. Dictionary as Range

(go to top)

In [3]:
```
world = {'iran':30.55, 'albania':2.77, 'algeria': 39.21}
```

In [10]:
```
for key, value in world.items():
    print(key + ':' +  str(value))
```

```
iran:30.55
albania:2.77
algeria:39.21
```

In [11]:
```
for country, population in world.items():
    print(country + ':' +  str(population))
```

```
iran:30.55
albania:2.77
algeria:39.21
```

5.5. Numpy Array as Range

(go to top)

In [13]:
```
import numpy as np
```

In [8]:
```
height = np.array([1.73, 1.68, 1.71, 1.89, 1.79])
weight = np.array([1.25, 1.23, 1.24, 1.29, 1.22])
```

In [11]:
```
height
#weight
```

Out[11]:
```
array([1.73, 1.68, 1.71, 1.89, 1.79])
```

In [13]:
```
for item in height:
    print(item)
```

```
1.73
1.68
1.71
1.89
1.79
```

- 2D Array

In [17]:
```
np_2d = np.array([height, weight])
np_2d
```

Out[17]:
```
array([[1.73, 1.68, 1.71, 1.89, 1.79],
       [1.25, 1.23, 1.24, 1.29, 1.22]])
```

In [18]:
```
for item in np_2d:
    print(item , '\n')
```

```
[1.73 1.68 1.71 1.89 1.79]

[1.25 1.23 1.24 1.29 1.22]
```

In [19]:
```
for item in np.nditer(np_2d):
    print(item)
```

```
1.73
1.68
1.71
1.89
1.79
1.25
1.23
1.24
1.29
1.22
```

In [12]:
```
for h, w in zip(height, weight):
    print(h,w)
```

```
1.73 1.25
1.68 1.23
1.71 1.24
1.89 1.29
1.79 1.22
```

5.6. DataFrame as Range

(go to top)

In [45]:
```
import pandas as pd

brics = pd.read_csv('datasets/brics.csv', index_col = 0)
display(brics)
```

	country	capital	area	population
BR	Brazil	Brasilia	8.516	200.40
RU	Russia	Moscow	17.100	143.50
IN	India	New Delhi	3.286	1252.00
CH	China	Beijing	9.597	1357.00
SA	South Africa	Pretoria	1.221	52.98

In [46]:
```
# Make index the first two letters of the entry in the country column

# brics.index = [entry[0:2] for entry in brics['country']]
# display(brics)
```

In [52]:
```
# print the column headers

for i in brics:
    print(i)
```

```
country
capital
area
population
```

In [64]:
```
data = brics.iterrows()
print(list(data))

# returns index, row data
```

```
[('BR', country         Brazil
capital       Brasilia
area             8.516
population       200.4
Name: BR, dtype: object), ('RU', country       Russia
capital       Moscow
area            17.1
population     143.5
Name: RU, dtype: object), ('IN', country          India
capital       New Delhi
area              3.286
population         1252
Name: IN, dtype: object), ('CH', country         China
capital       Beijing
area            9.597
population       1357
Name: CH, dtype: object), ('SA', country       South Africa
capital          Pretoria
area                1.221
population          52.98
Name: SA, dtype: object)]
```

In [65]:
```
for index, row in brics.iterrows():
    print(index + ":" + ['capital'])
```

```
BR:Brasilia
RU:Moscow
IN:New Delhi
CH:Beijing
SA:Pretoria
```

In [50]:
```
for index, row in brics.iterrows():
    print(index)
    print(row)
    print('\n')
```

```
BR
country         Brazil
capital       Brasilia
area             8.516
population       200.4
Name: BR, dtype: object


RU
country       Russia
capital       Moscow
area            17.1
population     143.5
Name: RU, dtype: object


IN
country           India
capital       New Delhi
area              3.286
population         1252
Name: IN, dtype: object


CH
country         China
capital       Beijing
area            9.597
population       1357
Name: CH, dtype: object


SA
country       South Africa
capital           Pretoria
area                 1.221
population           52.98
Name: SA, dtype: object
```

6. enumerate

- Recall that enumerate() returns an enumerate object that produces a sequence of tuples, and each of the tuples is an index-value pair.
- Use enumerate on a list
- iterables can be exhausted by making a list of them, you will have to redefine the iterable again if you want to perform more work.

(go to top)

In [2]:
```
avengers = ["hawkeye", "iron man", "thor", "quicksilver"]
```

In [25]:
```
e = enumerate(avengers)
print(list(e))
```

```
[(0, 'hawkeye'), (1, 'iron man'), (2, 'thor'), (3, 'quicksilver')]
```

In [3]:
```
e = enumerate(avengers, start = 10)
print(list(e))
```

```
[(10, 'hawkeye'), (11, 'iron man'), (12, 'thor'), (13, 'quicksilver')
]
```

In [4]:
```
list(e)
```

Out[4]:
```
[]
```

In [27]:
```
for index, value in enumerate(avengers, start = 1):
    print(index, value)
```

```
1 hawkeye
2 iron man
3 thor
4 quicksilver
```

In [67]:
```
fam = [1, 2, 3, 4, 5, 6, 7, 8, 9]

list(enumerate(fam))
```

Out[67]:
```
[(0, 1), (1, 2), (2, 3), (3, 4), (4, 5), (5, 6), (6, 7), (7, 8), (8, 9)]
```

In [68]:
```
for i, j in enumerate(fam):
    print('index' , i , ':', j)
```

```
index 0 : 1
index 1 : 2
index 2 : 3
index 3 : 4
index 4 : 5
index 5 : 6
index 6 : 7
index 7 : 8
index 8 : 9
```

---

7. zip()

- Turn iterables to tuples

(go to top)

In [71]:
```
avengers = ["hawkeye", "iron man", "thor", "quicksilver"]
names = ['barton', 'start', 'odinson', 'maximoff']

z = zip(avengers, names)

display(list(z))
```

```
[('hawkeye', 'barton'),
 ('iron man', 'start'),
 ('thor', 'odinson'),
 ('quicksilver', 'maximoff')]
```

- ∗ : Splat Operator

In [72]:
```
mutants = ['charles xavier', 'bobby drake', 'kurt wagner', 'max eisenh
powers =['telepathy', 'thermokinesis', 'teleportation', 'magnetokinesi

z1 = zip(mutants, powers)
display(*z1)
```

```
('charles xavier', 'telepathy')

('bobby drake', 'thermokinesis')

('kurt wagner', 'teleportation')

('max eisenhardt', 'magnetokinesis')

('kitty pryde', 'intangibility')
```

- using */ making it a list will exhause the elements in your iterator, you will have to recreate the zip object you defined if you want to use it again

In [73]:
```
# cannot print z1 again unless it is recreated
print(list(z1))
```

```
[]
```

In [74]:
```
# redefine z1
z1 = zip(mutants, powers)
display(list(z1))
```

```
[('charles xavier', 'telepathy'),
 ('bobby drake', 'thermokinesis'),
 ('kurt wagner', 'teleportation'),
 ('max eisenhardt', 'magnetokinesis'),
 ('kitty pryde', 'intangibility')]
```

- Run two loops Simultaneously

In [75]:
```
for z1, z2 in zip(mutants, powers):
    print(z1, ':', z2)
```

```
charles xavier : telepathy
bobby drake : thermokinesis
kurt wagner : teleportation
max eisenhardt : magnetokinesis
kitty pryde : intangibility
```

---

8. iter()

(go to top)

In [80]:
```
flash = ['jay garrick', 'barry allen', 'wally west', 'bart allen']

# Create an iterator for flash: superhero
superhero =  iter(flash)

# Print each item from the iterator
print(next(superhero))
print(next(superhero))
print(next(superhero))
print(next(superhero))
```

```
jay garrick
barry allen
wally west
bart allen
```

- ∗ : Splat Operator
- Does this only work with strings? why does (* superhero) not print

In [79]:
```
#print(*superhero)
```

In [5]:
```
word = 'data'
it = iter(word)
print(* it, end="")
```

```
d a t a
```

In [9]:
```
nums = [1, 2, 3, 4, 5]
print (* iter(nums), sep='')
```

```
12345
```

In [8]:
```
print?
```

9. While Loop

(go to top)

In [2]:
```
offset = -6
```

In [3]:
```
while offset != 0:
    print('correcting....')

    if offset > 0:
        offset -= 1
    else:
        offset +=1

    print(offset)
```

```
correcting....
-5
correcting....
-4
correcting....
-3
correcting....
-2
correcting....
-1
correcting....
0
```

10. Nested Loops

(go to top)

11. Boolean Logic

(go to top)

In [42]:
```
a = False
b = 5
c = 0
d = ""
e = " "
```

In [37]:
```
bool(c)
```

Out[37]: False

In [38]:
```
bool(b)
```

Out[38]: True

In [41]:
```
bool(d)
```

Out[41]: False

In [43]:
```
bool(e)
```

Out[43]: True

In [44]:
```
bool([])
```

Out[44]: False

In [46]:
```
bool([1,2,3])
```

Out[46]: True

In [12]:
```
a or True
```

Out[12]: True

In [47]:
```
a and True
```

Out[47]: False

In [13]:
```
( a or (b and c) ) == ( (a or b) and (a or c) )
```

Out[13]: True

In [9]:
```
( a and (b or c) ) == ( (a and b) or (a and c) )
```

Out[9]: True

In [14]:
```
(not(not a)) == a
```

Out[14]: True

In [15]:
```
(not(a or b)) == ((not a)and(not b))
```

Out[15]: True

11.1 String Logic

(go to top)

In [2]:
```
'CHRIS' < 'chris'
```

Out[2]: True

In [5]:
```
'chrir' < 'chris'
```

Out[5]: True

11.2. Operators

(go to top)

In [16]:
```
my_kitchen = 18.0
your_kitchen = 14.0
x = 8
y = 9
```

In [15]:
```
print(my_kitchen > 10 and my_kitchen < 18)
print(10 < my_kitchen < 18)
```

```
False
False
```

In [17]:
```
print(my_kitchen > 17 or my_kitchen < 14)
```

```
True
```

In [21]:
```
print(not(not(x < 3) and not(y > 14 or y > 10)))
```

```
False
```

11.3. Numpy

(go to top)

In [26]:
```
import numpy as np
bmi = np.array([21.852, 20.975, 21.75, 24.747, 21.441])
```

In [27]:
```
bmi > 21
```

Out[27]:
```
array([ True, False,  True,  True,  True])
```

In [28]:
```
bmi < 22
```

Out[28]:
```
array([ True,  True,  True, False,  True])
```

In [29]:
```
np.logical_and(bmi > 21, bmi < 22)
```

Out[29]:
```
array([ True, False,  True, False,  True])
```

In [30]:
```
bmi[np.logical_and(bmi > 21, bmi < 22)]
```

Out[30]:
```
array([21.852, 21.75 , 21.441])
```

11.4. Pandas

(go to top)

In [31]:
```
import pandas as pd
brics = pd.read_csv('datasets/brics.csv', index_col = 0)
display(brics)
```

	country	capital	area	population
BR	Brazil	Brasilia	8.516	200.40
RU	Russia	Moscow	17.100	143.50
IN	India	New Delhi	3.286	1252.00
CH	China	Beijing	9.597	1357.00
SA	South Africa	Pretoria	1.221	52.98

In [32]:
```
brics['area'] > 9
```

Out[32]:
```
BR    False
RU     True
IN    False
CH     True
SA    False
Name: area, dtype: bool
```

In [33]:
```
brics[brics['area'] > 9]
```

Out[33]:

	country	capital	area	population
RU	Russia	Moscow	17.100	143.5
CH	China	Beijing	9.597	1357.0

In [34]:
```
np.logical_and(brics['area'] > 8, brics['area'] < 10)
```

Out[34]:
```
BR     True
RU    False
IN    False
CH     True
SA    False
Name: area, dtype: bool
```

In [35]:
```
brics[np.logical_and(brics['area'] > 8, brics['area'] < 10)]
```

Out[35]:

	country	capital	area	population
BR	Brazil	Brasilia	8.516	200.4
CH	China	Beijing	9.597	1357.0

In [36]:
```
brics[np.logical_and(brics['area'] > 8, brics['population'] > 200)]
```

Out[36]:

	country	capital	area	population
BR	Brazil	Brasilia	8.516	200.4
CH	China	Beijing	9.597	1357.0

In [ ]:
```

```

12. Title

(go to top)

13. Title

(go to top)

14. Title

(go to top)

15. Title

(go to top)

16. Title

(go to top)

17. Title

(go to top)

18. Title

(go to top)

19. Title

(go to top)

20. Title

(go to top)

In [ ]:

In [ ]: