Table of contents

- 27. list: `+ & *`
- 28. list: `len()`
- 29. list: `sorted()`
- 30. list: `.sort()`
- 31. list: `extend()`
- 32. list: `append`
- 33. list: `in`
- 34. list: `list()`
- 35. list: `sum()`
- 36. list: `.index()`
- 37. list: `.set()`
- 38. list: `.reverse() / reversed()`
- 39. list: `insert(i,x)`
- 40. list: `count(x)`
- 41. list: `remove(x)`
- 42. list: `pop(i)`
- 43. list: `nested lists`
- 44. list: `with for loops`
- 44b. list: `listing iterables`
- 44c. list: `list comprehensions`
- 44d. list: `list generators`

- 45. dict: `create from list`
- 46. dict: `create from tuple, zip`
- 47. dict: `.keys(), .values()`
- 48. dict: `in`
- 50. dict: `del()`
- 51. dict: `.pop()`
- 52. dict: `.update()`
- 53. dict: `indexing`
- 54. dict: `update / replace`
- 55. dict: `nested dicts`

- 56.
- 57.
- 58.
- 59.
- 60.

- 61. functions: `definition`

- 62. functions: `multiple arguments`
- 63. functions: `python builtins`
- 64. functions: `return functions`
- 65. functions: `arguments`
- 66. functions: `args - Variable Length (Positional) Arguments`
- 67. functions: `**kwargs - Variable Length Keyword Arguments`
- 68. function: `scope`
- 69. functions: `nested functions`
- 69b. functions: `passing mutable parameters`

- 70. lambda functions: `definition`
- 71. lambda function: `map()`
- 72. lambda functions: `filter()`
- 73. lambda functions: `reduce()`

- 74. loops: `if -if statement`
- 75. loops: `if-else statements`
- 76. loops: `if-elif-else statements`
- 77. loops: `for loops`
- 78. loops: `enumerate`
- 79. loops: `zip()`
- 80. loops: `iter()`
- 81. loops: `while loop`

- 82. useful code: simultaneous assignment
- 83. useful code: `common imports`
- 84. useful code: `name = main`
- 84b. useful code: `* unpacking`
- 84c. useful code: `any()`
- 84d. useful code: `str.isalpha()/isdigit()/isalnum()/islower()/isupper()`
- 84e. useful code: `textwrap`
- 84f. useful code: `zip`
- 84g. useful code: `set`
- 84h. useful code: `set comprehensions`

- 85. print: `newline`
- 86. print: `end=''`
- 87. print: `sep`
- 88.

---

1. operators: `arithmetic`, `comparison`, `assignment`, `logical`

(go to top)

- ### `arithmetic operators`

In [18]:
```
print(10 + 5)

float1 = 13.65
float2 = 3.40
print(float1 + float2)

num = 20
flt = 10.5
print(num + flt)
```

```
15
17.05
30.5
```

In [19]:
```
print(10 - 5)

float1 = -18.678
float2 = 3.55
print(float1 - float2)

num = 20
flt = 10.5
print(num - flt)
```

```
5
-22.228
9.5
```

In [20]:
```
print(40 * 10)

float1 = 5.5
float2 = 4.5
print(float1 * float2)

print(10.2 * 3)
```

```
400
24.75
30.599999999999998
```

In [21]:
```
print(40 / 10)

float1 = 5.5
float2 = 4.5
print(float1 / float2)
print(12.4 / 2)
```

```
4.0
1.2222222222222223
6.2
```

In [22]:
```
print(43 // 10)

float1 = 5.5
float2 = 4.5
print(5.5 // 4.5)
print(12.4 // 2)
```

```
4
1.0
6.0
```

In [23]:
```
print(10 % 2)

twenty_eight = 28
print(twenty_eight % 10)

print(-28 % 10)  # The remainder is positive if the right-hand operand is po
print(28 % -10)  # The remainder is negative if the right-hand operand is ne
print(34.4 % 2.5)  # The remainder can be a float
```

```
0
8
2
-2
1.8999999999999986
```

In [24]:
```
# Different precedence
print(10 - 3 * 2)  # Multiplication computed first, followed by subtraction

# Same precedence
print(3 * 20 / 5)  # Multiplication computed first, followed by division
print(3 / 20 * 5)  # Division computed first, followed by multiplication
```

```
4
12.0
0.75
```

In [25]:
```
print((10 - 3) * 2)  # Subtraction occurs first
print((18 + 2) / (10 % 8))
```

```
14
10.0
```

In [1]:
```
a = 5
b = 2
```

In [2]:
```
a + b
```

Out[2]: 7

In [3]:
```
a - b
```

Out[3]: 3

In [4]:
```
a * b
```

Out[4]: 10

In [5]:
```
a / b
```

Out[5]: 2.5

In [7]:
```
a // b
```

Out[7]: 2

In [8]:
```
a ** b
```

Out[8]: 25

- ### comparison operators

In [26]:
```
num1 = 5
num2 = 10
num3 = 10
print(num2 > num1)  # 10 is greater than 5
print(num1 > num2)  # 5 is not greater than 10

print(num2 == num3)  # Both have the same value
print(num3 != num1)  # Both have different values

print(3 + 10 == 5 + 5)  # Both are not equal
print(3 <= 2)  # 3 is not less than or equal to 2
```

```
True
False
True
True
False
False
```

In [9]:
```
a == b
```

Out[9]:
```
False
```

In [10]:
```
a != b
```

Out[10]:
```
True
```

In [11]:
```
a < b
```

Out[11]:
```
False
```

In [12]:
```
a <= b
```

Out[12]:
```
False
```

In [13]:
```
a > b
```

Out[13]:
```
True
```

In [14]:
```
a is not b
```

Out[14]:
```
True
```

In [16]:
```
a is b
```

Out[16]:
```
False
```

- ### assignment operators

Let's go through a few examples to see how values are assigned to variables.

Variables are mutable, so we can change their values whenever we want!

One thing to note is that when a variable, first, is assigned to another variable, second, its value is copied into second.

Hence, if we later change the value of first, second will remain unaffected:

In [28]:
```
year = 2019
print(year)

year = 2020
print(year)

year = year + 1  # Using the existing value to create a new one
print(year)
```

```
2019
2020
2021
```

In [29]:
```
first = 20
second = first
first = 35  # Updating 'first'

print(first, second)  # 'second' remains unchanged
```

```
35 20
```

In [30]:
```
num = 10
print(num)

num += 5
print(num)

num -= 5
print(num)

num *= 2
print(num)

num /= 2
print(num)

num **= 2
print(num)

# Try all the others here!
```

```
10
15
10
20
10.0
100.0
```

- ### `logical operators`

In [ ]:
```
# OR Expression
my_bool = True or False
print(my_bool)

# AND Expression
my_bool = True and False
print(my_bool)

# NOT expression
my_bool = False
print(not my_bool)
```

In [31]:
```
print(10 * True)
print(10 * False)
```

```
10
0
```

In [ ]:
```
aaaa
```

In [17]:
```
c = -124
print(abs(c))
```

```
124
```

In [18]:
```
round(3.49)
```

Out[18]:
```
3
```

In [19]:
```
round(3.5)
```

Out[19]:
```
4
```

In [21]:
```
pi = 3.141592653589793
round(pi,2)
```

Out[21]:
```
3.14
```

In [22]:
```
round(pi,4)
```

Out[22]:
```
3.1416
```

In [23]:
```
max(a,b)
```

Out[23]:
```
5
```

In [24]:
```
max([1, 2, 5, 9])
```

Out[24]:
```
9
```

In [25]:
```
min(a,b)
```

Out[25]:
```
2
```

In [26]:
```
min([1, 2, 5, 9])
```

Out[26]:
```
1
```

In [28]:
```
test_list =  [1, 2, 5, 9]
test_tuple =  (1, 2, 5, 9)
sum(test_list)
```

Out[28]:
```
17
```

In [30]:
```
sum(test_tuple)
```

Out[30]:
```
17
```

In [31]:
```
average = sum(test_list) / len(test_list)
average
```

Out[31]:
```
4.25
```

In [34]:
```
# only works with iterables and numbers only
sum(2, 5)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-34-bb32f845c22d> in <module>
      1 # only works with iterables and numbers only
----> 2 sum(2, 5)

TypeError: 'int' object is not iterable
```

1.1 numbers: `complex numbers`

(go to top)

In [3]:
```
print(complex(10, 20))  # Represents the complex number (10 + 20j)
print(complex(2.5, -18.2))  # Represents the complex number (2.5 - 18.2j)

complex_1 = complex(0, 2)
complex_2 = complex(2, 0)
print(complex_1)
print(complex_2)
```

```
(10+20j)
(2.5-18.2j)
2j
(2+0j)
```

---

2. type conversion

(go to top)

In [1]:
```
i = '3.14159'
type(i)
```

Out[1]:
```
str
```

- `str to float`

In [36]:
```
j = float(i)

print(j)
type(j)
```

Out[36]:
```
3.14159
float
```

- `float to int`

In [37]:
```
k = int(j)

print(k)
type(k)
```

Out[37]:
```
3
int
```

- `str to int`

In [40]:
```
int('5')
```

Out[40]:
```
5
```

- `range to list`

In [43]:
```
m = range(10)
print(type(m))
```

```
<class 'range'>
```

In [44]:
```
n = list(m)
print(n)
print(type(n))
```

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
<class 'list'>
```

- `float to str`

In [45]:
```
o = str(j)
print(o)
print(type(o))
```

```
3.14159
<class 'str'>
```

- `str (num like decimal) to int`

In [42]:
```
int(i)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-42-e37c129aabaf> in <module>
----> 1 int(i)

ValueError: invalid literal for int() with base 10: '3.14159'
```

---

3. strings: in loops/sequences

(go to top)

In [48]:
```
message = "Random"

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

```
R
a
n
d
o
m
```

In [49]:
```
e = 'python'
list(e)
```

Out[49]:
```
['p', 'y', 't', 'h', 'o', 'n']
```

3.1 strings: `multi-line strings`

(go to top)

In [9]:
```
multiple_lines = '''Triple quotes allows
multi-line string.'''
print(multiple_lines)
```

```
Triple quotes allows
multi-line string.
```

3.2. strings: string immutability

(go to top)

Once we assign a value to a string, we can't update it later. How about verifying it with an executable below?

In [7]:
```
string = "Immutability"
string[0] = 'O' # Will give error
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
~\AppData\Local\Temp/ipykernel_25316/4127141596.py in <module>
      1 string = "Immutability"
----> 2 string[0] = 'O' # Will give error

TypeError: 'str' object does not support item assignment
```

The above code gives TypeError because Python doesn't support item assignment in case of strings.Remember, assigning a new value to string variable doesn't mean that you've changed the value. Let's verify it with the id() method below.

Notice, when we assign a new value to str1 (at line 4) its identity changes not the value.

In [10]:
```
str1 = "hello"
print(id(str1))

str1 = "bye"
print(id(str1))
```

```
2262916152176
2262921441200
```

3.3. strings: ascii vs unicode

(go to top)

`In Python 3.x, all strings are unicode. But, older versions of Python (Python 2.x) support only ASCII characters.`

`To use unicode in Python 2.x, preceding the string with a u is must`

In [12]:
```
string = u"This is unicode"
```

---

4. strings: with `input()`

(go to top)

In [51]:
```
x, y = input("Enter the coodrinates (x,y): ").split(',')
print(x,y)
```

```
Enter the coodrinates (x,y): 5,9
5 9
```

In [52]:
```
month, day, year = input("Enter date as mm/dd/yyyy: ").split('/')
print(month, day, year)
```

```
Enter date as mm/dd/yyyy: 03/17/2021
03 17 2021
```

---

5. string: Slicing & Indexing

(go to top)

In [6]:
```
batman = "Bruce Wayne"
print(batman[-1])  # Corresponds to batman[10]
print(batman[-5])  # Corresponds to batman[6]
```

```
e
W
```

In [13]:
```
my_string = "This is MY string!"
print(my_string[0:4]) # From the start till before the 4th index
print(my_string[1:7])
print(my_string[8:len(my_string)]) # From the 8th index till the end
```

```
This
his is
MY string!
```

In [15]:
```
my_string = "This is MY string!"
print(my_string[0:7])  # A step of 1
print(my_string[0:7:2])  # A step of 2
print(my_string[0:7:5])  # A step of 5
```

```
This is
Ti s
Ti
```

In [16]:
```
my_string = "This is MY string!"
print(my_string[13:2:-1]) # Take 1 step back each time
print(my_string[17:0:-2]) # Take 2 steps back. The opposite of what happens
```

```
rts YM si s
!nrsY ish
```

In [17]:
```
my_string = "This is MY string!"
print(my_string[:8])  # All the characters before 'M'
print(my_string[8:])  # All the characters starting from 'M'
print(my_string[:])  # The whole string
print(my_string[::-1])  # The whole string in reverse (step is -1)
```

```
This is
MY string!
This is MY string!
!gnirts YM si sihT
```

In [61]:
```
message = 'random message'
```

In [62]:
```
message[0]
```

Out[62]:
```
'r'
```

In [63]:
```
message[-1]
```

Out[63]:
```
'e'
```

In [64]:
```
message[:]
```

Out[64]:
```
'random message'
```

In [65]:
```
message[0:2]
```

Out[65]:
```
'ra'
```

In [66]:
```
message[:2]
```

Out[66]:
```
'ra'
```

In [67]:
```
print(message[0:13:1])
```

```
random messag
```

In [68]:
```
print(message[0:13:2])
```

```
rno esg
```

---

6. string: backslash/newline

(go to top)

In [69]:
```
f = 'this is a \n new line'
print(f)
```

```
this is a
 new line
```

In [70]:
```
g = 'how to add a backslash \\ to a string'
print(g)
```

```
how to add a backslash \ to a string
```

In [ ]:
```

```

---

7. string: raw format

- `If you have a string with a lot of backslashes and no special characters, you might find this a bit annoying. Fortunately you can preface the leading quote of the string with r, which means that the characters should be interpreted as is:`

(go to top)

In [71]:
```
h = r'this\has\no\special\xters'
print(h)
```

```
this\has\no\special\xters
```

In [ ]:
```

```

---

8. string: formatting/templating

(go to top)

In [11]:
```
template = '{0:0.3f} {1:s} are worth US${2:d}'
```

- `{0:0.2f}` means to format the first argument as a floating-point number with two decimal places
- `{1:s}` means to format the second argument as a string.
- `{2:d}` means to format the third argument as an exact integer.

In [13]:
```
template.format(4.5560, 'Argentine Pesos', 1)
```

Out[13]:
```
'4.556 Argentine Pesos are worth US$1'
```

In [14]:
```
"Hello {0}, you may have won ${1} {2}".format("dude", 100000, 'lottery' )
```

Out[14]:
```
'Hello dude, you may have won $100000 lottery'
```

In [76]:
```
"This int, {0:5}, was placed in a field of width 5".format(7)
```

Out[76]:
```
'This int,     7, was placed in a field of width 5'
```

In [77]:
```
'{0:0.02f}'.format(55569.56457)
```

Out[77]:
```
'55569.56'
```

In [78]:
```
'{0:10.02f}'.format(55.56956457)
```

Out[78]:
```
'     55.57'
```

In [87]:
```
'{0:10.02}'.format(55.56956457)
```

Out[87]:
```
'   5.6e+01'
```

In [88]:
```
'{0:10.03}'.format(55.56956457)
```

Out[88]:
```
'      55.6'
```

In [83]:
```
#if you have. a float result 1.5 but you want to express it as $1.50

'${0:0.02f}'.format(1.5)
```

Out[83]:
```
'$1.50'
```

In [90]:
```
'{0:f}'.format(55569)
```

Out[90]:
```
'55569.000000'
```

In [91]:
```
'{0:d}'.format(556)
```

Out[91]:
```
'556'
```

In [92]:
```
'{0:s}'.format('a string here')
```

Out[92]:
```
'a string here'
```

In [93]:
```
# {index: width.precision}

"Compare {0} and {0:15} || and  {0:0.4f} and {0:0.4} || and {0:0.04f} and {0
```

Out[93]:
```
'Compare 3.1 and             3.1 || and  3.1000 and 3.1 || and 3.1000 and 3.
1 || and 3.1000000000000000888'
```

---

8. string: `formatting in loops`

(go to top)

In [50]:
```
seq = [(1, 2, 3), (4, 5, 6), (7, 8, 9)]

# for the first loop around it's doing  a, b, c, = (1,2,3)
for a, b, c in seq:
    print('a={0}, b={1}, c={2}'.format(a, b, c))
```

```
a=1, b=2, c=3
a=4, b=5, c=6
a=7, b=8, c=9
```

8a. string: formatting using `f'{}'`

(go to top)

In [53]:
```
var = 'nothing'

print(f'vacuum is full of {var}')
```

```
vacuum is full of nothing
```

---

9. string: operators : `+`, `*`

(go to top)

In [94]:
```
message = 'Random Message'
text = 'Text Text'
```

In [99]:
```
concatenate = message + text
print(concatenate)
```

```
Random MessageText Text
```

In [33]:
```
first_half = "Bat"
second_half = "man"

full_name = first_half + second_half
print(full_name)
```

```
Batman
```

In [100…
```
repeat = message * 2
print(repeat)
```

```
Random MessageRandom Message
```

In [35]:
```
print("ha" * 3)
```

```
hahaha
```

In [32]:
```
print('a' < 'b')  # 'a' has a smaller Unicode value

house = "Gryffindor"
house_copy = "Gryffindor"

print(house == house_copy)

new_house = "Slytherin"

print(house == new_house)

print(new_house <= house)

print(new_house >= house)
```

```
True
True
False
False
True
```

In [36]:
```
random_string = "This is a random string"

print('of' in random_string)  # Check whether 'of' exists in randomString
print('random' in random_string)  # 'random' exists!
```

```
False
True
```

---

10. strings: `len()`

(go to top)

In [101…
```
message = 'Random Message'
text = 'Text Text'
```

In [102…
```
print(len(message))
```

```
14
```

---

11. strings: `.split()`

(go to top)

In [110…
```
message = 'Random Message'
text = 'Text Text'
words = 'JUST TEXT IN UPPER CASE'
```

In [111…
```
student_scores = 'temi 50 60 90'
name, *scores = student_scores.split()
```

In [112…
```
print(name)
print(scores)
print(type(scores))
```

```
temi
['50', '60', '90']
<class 'list'>
```

In [113…
```
split_msg = message.split()
print(split_msg)
```

```
['Random', 'Message']
```

In [114…
```
split_msg = words.split(' ')
print(split_msg)
```

```
['JUST', 'TEXT', 'IN', 'UPPER', 'CASE']
```

In [115…
```
split_msg = words.split('T')
print(split_msg)
```

```
['JUS', ' ', 'EX', ' IN UPPER CASE']
```

`.split ("\t") for a tab seperated file line`
`file.readline().split('\t')`

---

12. strings : `.capitalise()`

(go to top)

In [3]:
```
message = 'random message'
text = 'Text Text'
words = 'JUST TEXT IN UPPER CASE'
```

In [4]:
```
print(message.capitalize())
print(words.capitalize())
```

```
Random message
Just text in upper case
```

---

13. strings: `.upper()` and `.lower()`

(go to top)

In [5]:
```
message = 'random message'
text = 'Text Text'
words = 'JUST TEXT IN UPPER CASE'
```

In [6]:
```
print(message.upper())
print(words.lower())
```

```
RANDOM MESSAGE
just text in upper case
```

In [7]:
```
print(message)
print(message.isupper())
print(message.islower())
```

```
random message
False
True
```

---

14. strings: `swapcase()`

(go to top)

In [8]:
```
message = 'random message'
text = 'Text Text'
words = 'JUST TEXT IN UPPER CASE'
```

In [10]:
```
print(message)
print(message.swapcase())
print(text)
print(text.swapcase())
```

```
random message
RANDOM MESSAGE
Text Text
tEXT tEXT
```

---

15. strings: `.title()`

(go to top)

In [11]:
```
message = 'random message'
text = 'Text Text'
words = 'JUST TEXT IN UPPER CASE'
```

In [12]:
```
print(words.title())
print(message.title())
```

```
Just Text In Upper Case
Random Message
```

---

16. strings: `in` and `not in`

(go to top)

In [13]:
```
message = 'random message'
text = 'Text Text'
words = 'JUST TEXT IN UPPER CASE'
```

In [14]:
```
print('R' in message)
print('R' not in message)
```

```
False
True
```

---

17. strings: `.count()`

(go to top)

In [15]:
```
message = 'random message'
text = 'Text Text'
words = 'JUST TEXT IN UPPER CASE'
```

In [16]:
```
print(message.count('s'))
print(words.count('T'))
```

```
2
3
```

---

18. strings: `.replace()`

(go to top)

In [13]:
```
message = 'random message'
text = 'Text Text'
words = 'JUST TEXT IN UPPER CASE'
```

In [17]:
```
print(message.replace('s','P'))
print(words.replace('T','S'))
```

```
random mePPage
JUSS SEXS IN UPPER CASE
```

---

19. strings: `.center()`

(go to top)

In [13]:
```
message = 'random message'
text = 'Text Text'
words = 'JUST TEXT IN UPPER CASE'
```

In [18]:
```
message.center(25)
```

Out[18]:
```
'      random message     '
```

In [19]:
```
message.center(35)
```

Out[19]:
```
'           random message          '
```

---

20. strings: `.ljust()` & `.rjust()`

(go to top)

In [13]:
```
message = 'random message'
text = 'Text Text'
words = 'JUST TEXT IN UPPER CASE'
```

In [23]:
```
print(len(message))
```

```
14
```

In [25]:
```
message.ljust(19)
```

Out[25]:
```
'random message     '
```

In [26]:
```
message.rjust(19)
```

Out[26]:
```
'     random message'
```

---

21. strings: `.find()`

(go to top)

In [27]:
```
message
```

Out[27]:
```
'random message'
```

In [29]:
```
message.find('r')
```

Out[29]:
```
0
```

In [30]:
```
message.find('a')
```

Out[30]:
```
1
```

In [31]:
```
message.find('n')
```

Out[31]:
```
2
```

---

22. strings: `.rfind()`

- returns the right-most position

(go to top)

In [34]:
```
message
```

Out[34]:
```
'random message'
```

In [35]:
```
message.find('a')
```

Out[35]:
```
1
```

In [36]:
```
message.rfind('a')
```

Out[36]:
```
11
```

In [ ]:
```

```

---

23. strings: `.join()`

(go to top)

In [37]:
```
message
```

Out[37]:
```
'random message'
```

In [39]:
```
" ".join('space between')
```

Out[39]:
```
's p a c e   b e t w e e n'
```

In [40]:
```
"||".join(['pipes',  'between', 'each', 'list', 'item'])
```

Out[40]:
```
'pipes||between||each||list||item'
```

- `list to string`

In [41]:
```
" ".join(['pipes',  'between', 'each', 'list', 'item'])
```

Out[41]:
```
'pipes between each list item'
```

- `list to string`

In [41]:
```
" ".join(['pipes',  'between', 'each', 'list', 'item'])
```

Out[41]:
```
'pipes between each list item'
```

- `examples`

In [51]:
```
lst = []
for i in range(1,int(input())+1):
    lst.append(str(i))
    print(''.join(lst[:(i-1)]) + ''.join(lst[::-1]), )
```

```
5
1
121
12321
1234321
123454321
```

---

24. strings: `lstrip()` & `rstrip()` & `.strip()`

- `copies the string with leading white pace removed`

(go to top)

- `lstrip`

In [42]:
```
message_2
```

Out[42]:
```
'    random message'
```

In [43]:
```
message_2.lstrip()
```

Out[43]:
```
'random message'
```

In [44]:
```
x = message_2.lstrip()

print("this is a", x)
```

```
this is a random message
```

- `rstrip`

In [45]:
```
message_3 = 'random message        '
message_3
```

Out[45]:
```
'random message        '
```

In [46]:
```
message_3.rstrip()
```

Out[46]:
```
'random message'
```

In [47]:
```
y = message_3.rstrip()

print(y, 'has no space')
```

```
random message has no space
```

- `strip`

In [48]:
```
s = '   2 5 6 8 6 4 9 3   '
s.strip()
```

Out[48]:
```
'2 5 6 8 6 4 9 3'
```

In [49]:
```
s = 'd d u j i 2 5 6 8 6 4 9 3   '
s.strip('d d u j i')
```

Out[49]:
```
'2 5 6 8 6 4 9 3'
```

In [50]:
```
txt = ",,,,,rrttgg.....banana....rrr"
txt.strip(",.grt")
```

Out[50]:
```
'banana'
```

---

25. input: `* unpacking`

(go to top)

In [53]:
```
values = 1,2,3,4,5
a, b, *rest = values
```

In [54]:
```
print(a,b)
```

```
1 2
```

In [55]:
```
rest
```

Out[55]:
```
[3, 4, 5]
```

25a. input: `map`

(go to top)

`convert input to integers`

In [29]:
```
# enter floats seprated by space 3 5 6
arr = map(int, input().split())

a = list(arr)
print(a)
print(a[0])
type(a[0])
```

```
3 5 6
[3, 5, 6]
3
```
Out[29]:
```
int
```

In [32]:
```
# Enter your code here. Read input from STDIN. Print output to STDOUT
n, m = map(int, input().split())
print(n)
print(m)
```

```
6 87
6
87
```

convert input to floats

In [30]:
```
# enter floats seprated by space 2.5 6.7 8
arr = map(float, input().split())

a = list(arr)
print(a)
print(a[0])
type(a[0])
```

```
2.5 6.7 8
[2.5, 6.7, 8.0]
2.5
```
Out[30]:
```
float
```

25b. input: `split()`

(go to top)

In [28]:
```
x = input('Enter integers sperated by space: ').split()
x
```

```
Enter integers sperated by space: 2 3 4 5
```
Out[28]:
```
['2', '3', '4', '5']
```

In [29]:
```
x = list(map(int, input('Enter integers sperated by space: ').split()))
x
```

```
Enter integers sperated by space: 2 3 4 5
```

Out[29]:
```
[2, 3, 4, 5]
```

In [38]:
```
y = 'text me'
b, *c = y.split()
print(b, c)
```

```
text ['me']
```

---

26. list: splicing & indexing

(go to top)

In [57]:
```
list1 = ['a', 'b', 'c', 'd', 'e']
```

In [58]:
```
list1[0]
```

Out[58]:
```
'a'
```

In [59]:
```
list1[0]
```

Out[59]:
```
'e'
```

In [61]:
```
list1[0] = 'z'
list1
```

Out[61]:
```
['z', 'b', 'c', 'd', 'e']
```

In [62]:
```
list1[0:3]
```

Out[62]:
```
['z', 'b', 'c']
```

In [63]:
```
list1[0:3] = 'ayz'
list1
```

Out[63]:
```
['a', 'y', 'z', 'd', 'e']
```

In [64]:
```
seq =[7,2,3,7,5,6,0,1]
seq[3:4] = [6, 3]
seq
```

Out[64]:
```
[7, 2, 3, 6, 3, 5, 6, 0, 1]
```

- `reverse slicing`

In [65]:
```
lst = ['1', '2', '3', '4', '5']
```

In [66]:
```
lst[5::-1]
```

Out[66]:
```
['5', '4', '3', '2', '1']
```

In [67]:
```
lst[::-1]
```

Out[67]:
```
['5', '4', '3', '2', '1']
```

---

27. list: + & *

(go to top)

In [68]:
```
list1 = ['a', 'bbbbb', 'ca', 'deb', 'e']
list2 = ['f', 'g', 'g', 'i', 'j']
list5 = [5,6,9,8,2,1,6,3,4,0]
```

In [69]:
```
list1 + list2
```

Out[69]:
```
['a', 'bbbbb', 'ca', 'deb', 'e', 'f', 'g', 'g', 'i', 'j']
```

In [70]:
```
list1 * 2
```

Out[70]:
```
['a', 'bbbbb', 'ca', 'deb', 'e', 'a', 'bbbbb', 'ca', 'deb', 'e']
```

---

28. list: `len()`

(go to top)

In [71]:
```
list1 = ['a', 'ca', 'deb', 'e', 'bbbbb',]
list2 = ['f', 'g', 'g', 'i', 'j']

list3 = ['a', 'b', 2, 'd', 'e']
list4 = ['f', 'g', 45, 'i', 'j']
list5 = [5,6,9,8,2,1,6,3,4,0]
```

In [72]:
```
len(list4)
```

Out[72]:
```
5
```

---

29. list: `sorted()`

(go to top)

In [73]:
```
list1 = ['a', 'ca', 'deb', 'e', 'bbbbb',]
list2 = ['f', 'g', 'g', 'i', 'j']

list3 = ['a', 'b', 2, 'd', 'e']
list4 = ['f', 'g', 45, 'i', 'j']
list5 = [5,6,9,8,2,1,6,3,4,0]
```

In [74]:
```
sorted(list5)
```

Out[74]:
```
[0, 1, 2, 3, 4, 5, 6, 6, 8, 9]
```

In [75]:
```
sorted(list1)
```

Out[75]:
```
['a', 'bbbbb', 'ca', 'deb', 'e']
```

---

30. list: `.sort()

- `list.sort(reverse=True|False, key=myFunc)`
- `#list5.sort() #alphabetical order if strings`
- `#list5.sort(key=int)`

(go to top)

In [39]:
```
list1 = ['a', 'ca', 'deb', 'e', 'bbbbb',]
list2 = ['f', 'g', 'g', 'i', 'j']

list3 = ['a', 'b', 2, 'd', 'e']
list4 = ['f', 'g', 45, 'i', 'j']
list5 = [5,6,9,8,2,1,6,3,4,0]
```

In [40]:
```
list5.sort()
list5
```

Out[40]:
```
[0, 1, 2, 3, 4, 5, 6, 6, 8, 9]
```

In [41]:
```
list1.sort()
list1
```

Out[41]:
```
['a', 'bbbbb', 'ca', 'deb', 'e']
```

- `reverse=True`

In [42]:
```
list5.sort(reverse=True)
list5
```

Out[42]:
```
[9, 8, 6, 6, 5, 4, 3, 2, 1, 0]
```

In [43]:
```
list1.sort(reverse=True)
list1
```

Out[43]:
```
['e', 'deb', 'ca', 'bbbbb', 'a']
```

- `key = len`

In [44]:
```
list1.sort(key=len)
list1
```

Out[44]:
```
['e', 'a', 'ca', 'deb', 'bbbbb']
```

In [45]:
```
list1.sort(reverse=True, key=len)
list1
```

Out[45]:
```
['bbbbb', 'deb', 'ca', 'e', 'a']
```

- `key = int`

In [46]:
```
list5.sort(reverse=True, key=int)
list5
```

Out[46]:
```
[9, 8, 6, 6, 5, 4, 3, 2, 1, 0]
```

---

31. list: `extend()`

(go to top)

In [47]:
```
print(list3)
print(list4)

print('')
list3.extend('xyz')
print(list3)

print('')
list4.extend(list1)
print(list4)
```

```
['a', 'b', 2, 'd', 'e']
['f', 'g', 45, 'i', 'j']

['a', 'b', 2, 'd', 'e', 'x', 'y', 'z']

['f', 'g', 45, 'i', 'j', 'bbbbb', 'deb', 'ca', 'e', 'a']
```

In [ ]:
```

```

---

32. list: append

(go to top)

In [48]:
```
print(list1)
print(list2)

print('')
list1.append('abc')
print(list1)

print('')
list2.append(list1)
print(list2)
```

```
['bbbbb', 'deb', 'ca', 'e', 'a']
['f', 'g', 'g', 'i', 'j']

['bbbbb', 'deb', 'ca', 'e', 'a', 'abc']

['f', 'g', 'g', 'i', 'j', ['bbbbb', 'deb', 'ca', 'e', 'a', 'abc']]
```

---

33. list: `in`

(go to top)

In [50]:
```
print(list1)

'a' in list1
```

```
['bbbbb', 'deb', 'ca', 'e', 'a', 'abc']
```

Out[50]:
```
True
```

---

34. list: `list()`

(go to top)

In [52]:
```
type({'Test', 'Math', 1, 3, 'Five'})
```

Out[52]:
```
set
```

In [53]:
```
type(list({'Test', 'Math', 1, 3, 'Five'}))
```

Out[53]:
```
list
```

---

35. list: `sum()`

(go to top)

In [54]:
```
list5
```

Out[54]:
```
[9, 8, 6, 6, 5, 4, 3, 2, 1, 0]
```

In [55]:
```
sum(list5)
```

Out[55]:
```
44
```

---

36. list: `.index()`

(go to top)

- `The index() method returns an integer that represents the index of first match of specified element in the List.`
- `list_name.index(element, start, end)`
  - element - The element whose lowest index will be returned.
  - start (Optional) - The position from where the search begins.
  - end (Optional) - The position from where the search ends.

In [56]:
```
list5
```

Out[56]:
```
[9, 8, 6, 6, 5, 4, 3, 2, 1, 0]
```

In [57]:
```
list5.index(6)
```

Out[57]:
```
2
```

In [58]:
```
list5.index(6,3, -1)
```

Out[58]:
```
3
```

---

37. list: `set()`

(go to top)

In [59]:
```
lst = ['apple', 'banana', 'apple', 'orange']
lst_2 = set(lst)
```

In [60]:
```
lst_2
```

Out[60]:
```
{'apple', 'banana', 'orange'}
```

---

38. list: `.reverse() / reversed()`

(go to top)

In [66]:
```
lst = ['apple', 'banana', 'apple', 'orange']
lst
```

Out[66]:
```
['apple', 'banana', 'apple', 'orange']
```

In [67]:
```
lst.reverse()
lst
```

Out[67]:
```
['orange', 'apple', 'banana', 'apple']
```

In [68]:
```
list(reversed(lst))
```

Out[68]:
```
['apple', 'banana', 'apple', 'orange']
```

---

39. list: `insert(i,x)`

(go to top)

- `Inserts x into list at index i.`

In [71]:
```
lst = ['apple', 'banana', 'apple', 'orange']
lst
```

Out[71]:
```
['apple', 'banana', 'apple', 'orange']
```

In [72]:
```
lst.insert(0, 'guava')
lst
```

Out[72]:
```
['guava', 'apple', 'banana', 'apple', 'orange']
```

---

40. list: `count(x)`

(go to top)

- `Returns the number of occurrences of x in list.`

In [73]:
```
lst = ['apple', 'banana', 'apple', 'orange']
lst
```

In [74]:
```
lst.count('apple')
```

Out[74]:
```
2
```

---

41. list: `remove(x)`

(go to top)

- `Deletes the first occurrence of x in list.`

In [75]:
```
lst = ['apple', 'banana', 'apple', 'orange']
lst
```

Out[75]:
```
['apple', 'banana', 'apple', 'orange']
```

In [76]:
```
lst.remove('apple')
lst
```

Out[76]:
```
['banana', 'apple', 'orange']
```

---

42. list: `pop(i)`

(go to top)

- `Deletes the ith element of the list and returns its value.`

In [78]:
```
lst = ['apple', 'banana', 'apple', 'orange']
lst
```

Out[78]:
```
['apple', 'banana', 'apple', 'orange']
```

In [79]:
```
lst.pop(2)
```

Out[79]:
```
'apple'
```

In [80]:
```
lst
```

Out[80]:
```
['apple', 'banana', 'orange']
```

In [81]:
```
lst.pop(-1)
lst
```

Out[81]:
```
['apple', 'banana']
```

---

43. list: `nested lists`

(go to top)

In [82]:
```
house = [['hallway', 11.25],['kitchen', 18.0], ['kiving room', 20.0], ['bedr
house
```

Out[82]:
```
[['hallway', 11.25],
 ['kitchen', 18.0],
 ['kiving room', 20.0],
 ['bedroom', 10.75],
 ['bathroom', 9.5]]
```

---

44. list: `with for loops`

(go to top)

In [84]:
```
list1 = ['a', 'b', 'c', 'd', 'e']

for entry in list1:
    print(entry)
```

```
a
b
c
d
e
```

.append()

In [88]:
```
squares = []
numbers = []

for i in range(1,10):
    squares.append(i**2)
    numbers.append(i)

print(numbers, '\n', squares)
```

```
[1, 2, 3, 4, 5, 6, 7, 8, 9]
 [1, 4, 9, 16, 25, 36, 49, 64, 81]
```

nested list

In [89]:
```
house
```

Out[89]:
```
[['hallway', 11.25],
 ['kitchen', 18.0],
 ['kiving room', 20.0],
 ['bedroom', 10.75],
 ['bathroom', 9.5]]
```

In [90]:
```
for i in house:
    print(i)
```

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

In [91]:
```
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
```

44b. list: `listing iterables`

(go to top)

In [1]:
```
list(range(10))
```

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

In [ ]:
```
list('man')
```

44c. list: `list comprehensions`

(go to top)

- `A quicker way to create lists from any iterable (list, range, strings)`

In [1]:
```
numbers = [1,2,3,4,5]

# Make a new list which contains all of the item of numbers +1
new_nums = [nums +1 for nums in numbers]
print(new_nums)
```

```
[2, 3, 4, 5, 6]
```

In [2]:
```
result = [num for num in range(11)]
print(result)
```

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

In [3]:
```
x = [letter for letter in "hey"]
print(x)
```

```
['h', 'e', 'y']
```

In [3]:
```
strings = ['a', 'as', 'bat', 'car', 'dove', 'python']
[x.upper() for x in strings if len(x) > 2]
```

Out[3]:
```
['BAT', 'CAR', 'DOVE', 'PYTHON']
```

---

nested for loop list comprehensions

In [29]:
```
matrix_1 = [row for col in range(0,5) for row in range(5,10)]
display(matrix_1)
```

```
[5, 6, 7, 8, 9, 5, 6, 7, 8, 9, 5, 6, 7, 8, 9, 5, 6, 7, 8, 9, 5, 6, 7, 8, 9]
```

In [31]:
```
matrix_1 = [[col for col in range(5,10)] for row in range(0,5)]
display(matrix_1)

# Print the matrix
for row in matrix_1:
    print(row)
```

```
[[5, 6, 7, 8, 9],
 [5, 6, 7, 8, 9],
 [5, 6, 7, 8, 9],
 [5, 6, 7, 8, 9],
 [5, 6, 7, 8, 9]]
[5, 6, 7, 8, 9]
[5, 6, 7, 8, 9]
[5, 6, 7, 8, 9]
[5, 6, 7, 8, 9]
[5, 6, 7, 8, 9]
```

In [16]:
```
matrix = []

for num in range(0,5):
    row = []
    for col in range(5,10):
        row.append(col)
    matrix.append(row)

display(matrix)

# Print the matrix
for row in matrix:
    print(row)
```

```
[[5, 6, 7, 8, 9],
 [5, 6, 7, 8, 9],
 [5, 6, 7, 8, 9],
 [5, 6, 7, 8, 9],
 [5, 6, 7, 8, 9]]
[5, 6, 7, 8, 9]
[5, 6, 7, 8, 9]
[5, 6, 7, 8, 9]
[5, 6, 7, 8, 9]
[5, 6, 7, 8, 9]
```

In [7]:
```
pairs2 = [(num3, num4) for num3 in range(0,2) for num4 in range(6, 8)]
print(pairs2)
```

```
[(0, 6), (0, 7), (1, 6), (1, 7)]
```

In [9]:
```
pairs2 = [(num3, num4) for num4 in range(6, 8) for num3 in range(0,2) ]
print(pairs2)
```

```
[(0, 6), (1, 6), (0, 7), (1, 7)]
```

---

conditionals as list comprehensions

In [15]:
```
y = [x ** 2 for x in range(10) if x % 2 == 0]
print(y)
```

```
[0, 4, 16, 36, 64]
```

In [1]:
```
y = [x ** 2 if x % 2 == 0 else 0 for x in range(10)]
print(y)
```

```
[0, 0, 4, 0, 16, 0, 36, 0, 64, 0]
```

- In the output expression, keep the string as-is if the number of characters is >= 7, else replace it with an empty string

In [7]:
```
# Create a list of strings: fellowship
fellowship = ['frodo', 'samwise', 'merry', 'aragorn', 'legolas', 'boromir',

# Create list comprehension: new_fellowship with strings with 7 characters c
new_fellowship = [member if len(member) >= 7 else '' for member in fellowshi

# Print the new list
print(new_fellowship)
```

```
['', 'samwise', '', 'aragorn', 'legolas', 'boromir', '']
```

In [14]:
```
# Create list comprehension: new_fellowship with strings with 7 characters c
new_fellowship = [member for member in fellowship if len(member) >= 7]

# Print the new list
print(new_fellowship)
```

```
['samwise', 'aragorn', 'legolas', 'boromir']
```

---

dictionary comprehensions

- Curly braces `{}` instead of `[]`
- Key and value are seperated by a colon in the output expression

In [19]:
```
# Create a list of strings: fellowship
fellowship = ['frodo', 'samwise', 'merry', 'aragorn', 'legolas', 'boromir',

# Create dict comprehension: new_fellowship
new_fellowship = {member: len(member) for member in fellowship}

# Print the new dictionary
print(new_fellowship)
```

```
{'frodo': 5, 'samwise': 7, 'merry': 5, 'aragorn': 7, 'legolas': 7, 'boromir'
: 7, 'gimli': 5}
```

In [20]:
```
pos_neg = {num: -num for num in range(9)}
display(pos_neg)
```

```
{0: 0, 1: -1, 2: -2, 3: -3, 4: -4, 5: -5, 6: -6, 7: -7, 8: -8}
```

---

comprehensions with zip

In [22]:
```
lists = [(x, y) for x, y in zip(range(1,11), range(11,21))]
print(lists)
```

```
[(1, 11), (2, 12), (3, 13), (4, 14), (5, 15), (6, 16), (7, 17), (8, 18), (9,
19), (10, 20)]
```

44d. list: list generators

(go to top)

- Generators take up less memory, for large iterations use generators
- use ( ) not [ ]

In [1]:
```
list2 = (x for x in range(10))
list3 = (x for x in range(10))
```

- The list is generated when it is needed as follows

In [4]:
```
for x in list2:
    print(x)
```

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

In [7]:
```
print(next(list3))
print(next(list3))
```

```
0
1
```

I think

- You cannot run the for loop and the next() funtion without re-generating the geenrator
- Because the generator doesn't actually construct a list once you run a for loop through all its values there will be nothing left
- in which case you will get
- StopIteration exception is raised when there are no elements left to call.

In [8]:
```
# Create generator object: result
result = (num for num in range(0,31))

# Print the first 5 (0-4) values
print(next(result))
print(next(result))
print(next(result))
print(next(result))
print(next(result), '\n')

# Print the rest(5-30)  of the values. you can see that is starts from 5 and
for value in result:
    print(value)
```

```
0
1
2
3
4

5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
```

In [2]:
```python
list4 = (digits for digits in range(10))

gen_list = list(list4)
print(gen_list)
```

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

---

generators and memory

- compare the following
- the comprehension takes forever to compute (i literally hear my pc fan getting louder)
- while the generator is created instantly

In [12]:
```python
# DO NOT RUN THIS. LEAVE AS COMMENT iF YOU DO YOUR PC WILL FREEZE

# count = [num for num in range(10 ** 1000000)]
# print(count)
```

In [10]:
```
count_gen = (num for num in range(10 ** 1000000))
print(next(count_gen))
print(next(count_gen))
```

```
0
1
```

same rules as constructors

In [20]:
```
even = (num for num in range(1,10) if num % 2 == 0)
print(list(even))
```

```
[2, 4, 6, 8]
```

In [21]:
```
# Create a list of strings: lannister
lannister = ['cersei', 'jaime', 'tywin', 'tyrion', 'joffrey']
```

In [22]:
```
# Create a generator object: lengths
lengths = (len(person) for person in lannister)

# Iterate over and print the values in lengths
for value in lengths:
    print(value)
```

```
6
5
5
6
7
```

Generator Functions

- They are defined like regular functions with def:
- The dont use keyword `return` they use `yield`
- They yield sequence of values instead of returning a single value

In [1]:
```
def num_sequence(n):
    """Generates values from 0 to n"""
    i = 0
    while n > i:
        yield i
        i += 1
```

In [2]:
```
print(num_sequence(5))
```

```
<generator object num_sequence at 0x7fd583774890>
```

In [4]:
```
for i in num_sequence(5):
    print(i)
```

```
0
1
2
3
4
```

44e. list: `generator unpacking with * and sep`

(go to top)

In [12]:
```
count_gen = (num for num in range(10 ** 1))
print(next(count_gen))
print(next(count_gen))
```

```
0
1
```

In [14]:
```
count_gen = (num for num in range(10 ** 1))
print(* count_gen)
```

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

In [25]:
```
count_gen = (num for num in range(10 ** 1))
print(* count_gen, sep='-')
```

```
0-1-2-3-4-5-6-7-8-9
```

In [24]:
```
count_gen = (num for num in range(10 ** 1))
print(* count_gen, sep='')
```

```
0123456789
```

44f. list: `quick repating list`

(go to top)

In [36]:
```
dice = [6]*5
dice
```

Out[36]:
```
[6, 6, 6, 6, 6]
```

---

45. dict: `create from list`

(go to top)

In [129…
```
pop = [30.55, 2.77, 39.21, 25.61, 36.52]
countries = ['Afghanistan', 'Albania', 'Algeria', 'Nigeria', 'Ghana']
```

In [130…
```
world = {country: pop for country, pop in zip(countries, pop)}
print(world)
```

```
{'Afghanistan': 30.55, 'Albania': 2.77, 'Algeria': 39.21, 'Nigeria': 25.61,
'Ghana': 36.52}
```

---

46. dict: `create from tuple, zip`

(go to top)

In [131…
```
first_names = ('Nolan', 'Roger', 'Curt')
last_names = ('Ryan', 'Clemens', 'Schilling')
```

In [132…
```
mapping = {key: value for key, value in zip(first_names, last_names)}

mapping
```

Out[132]:
```
{'Nolan': 'Ryan', 'Roger': 'Clemens', 'Curt': 'Schilling'}
```

In [133…
```
dict(zip(range(5), reversed(range(5))))
```

Out[133]:
```
{0: 4, 1: 3, 2: 2, 3: 1, 4: 0}
```

---

47. dict: `.keys()`, `.values()`

(go to top)

In [134…
```
world
```

Out[134]:
```
{'Afghanistan': 30.55,
 'Albania': 2.77,
 'Algeria': 39.21,
 'Nigeria': 25.61,
 'Ghana': 36.52}
```

In [135…
```
world.keys()
```

Out[135]:
```
dict_keys(['Afghanistan', 'Albania', 'Algeria', 'Nigeria', 'Ghana'])
```

In [136…
```
world.values()
```

Out[136]:
```
dict_values([30.55, 2.77, 39.21, 25.61, 36.52])
```

In [137…
```
list(world.values())
```

Out[137]:
```
[30.55, 2.77, 39.21, 25.61, 36.52]
```

---

48. dict: `in`

(go to top)

In [138…
```
world
```

Out[138]:
```
{'Afghanistan': 30.55,
 'Albania': 2.77,
 'Algeria': 39.21,
 'Nigeria': 25.61,
 'Ghana': 36.52}
```

In [139…
```
'Nigeria' in world
```

Out[139]:
```
True
```

---

50. dict: `del()`

(go to top)

In [140…
```
world
```

Out[140]:
```
{'Afghanistan': 30.55,
 'Albania': 2.77,
 'Algeria': 39.21,
 'Nigeria': 25.61,
 'Ghana': 36.52}
```

In [141…
```
del(world['Ghana'])
print(world)
```

```
{'Afghanistan': 30.55, 'Albania': 2.77, 'Algeria': 39.21, 'Nigeria': 25.61}
```

---

51. dict: `pop()`

(go to top)

In [142…
```
world
```

Out[142]:
```
{'Afghanistan': 30.55, 'Albania': 2.77, 'Algeria': 39.21, 'Nigeria': 25.61}
```

In [143…
```
world.pop('Albania')
world
```

Out[143]:
```
{'Afghanistan': 30.55, 'Algeria': 39.21, 'Nigeria': 25.61}
```

---

52. dict: `update()`

(go to top)

In [144…
```
world
```

Out[144]:
```
{'Afghanistan': 30.55, 'Algeria': 39.21, 'Nigeria': 25.61}
```

In [145…
```
world.update({'Nigeria': 23.2, 'Ghana': 45.7})
world
```

Out[145]:
```
{'Afghanistan': 30.55, 'Algeria': 39.21, 'Nigeria': 23.2, 'Ghana': 45.7}
```

---

53. dict: `indexing`

(go to top)

In [146…
```
world
```

Out[146]:
```
{'Afghanistan': 30.55, 'Algeria': 39.21, 'Nigeria': 23.2, 'Ghana': 45.7}
```

In [147…
```
world['Nigeria']
```

Out[147]:
```
23.2
```

---

54. dict: `update/replace`

(go to top)

In [148…
```
world
```

Out[148]:
```
{'Afghanistan': 30.55, 'Algeria': 39.21, 'Nigeria': 23.2, 'Ghana': 45.7}
```

In [152…
```
world['Ethiopia'] = 24.25
world
```

Out[152]:
```
{'Afghanistan': 30.55,
 'Algeria': 39.21,
 'Nigeria': 66.32,
 'Ghana': 45.7,
 'Ethiopia': 24.25}
```

In [151…
```
world['Nigeria'] = 66.32
world
```

Out[151]:
```
{'Afghanistan': 30.55,
 'Algeria': 39.21,
 'Nigeria': 66.32,
 'Ghana': 45.7,
 'Ethiopia': 24.25}
```

---

55. dict: `nested dicts`

(go to top)

In [153…
```
europe =  {'spain': {'capital': 'madrid', 'population': 47.66 },
           'germany': {'capital': 'berlin', 'population': 23.66 },
           'nigeria': {'capital': 'lagos', 'population': 34.66 },
           'usa': {'capital': 'washington', 'population': 97.66 }}
```

In [154…
```
europe
```

Out[154]:
```
{'spain': {'capital': 'madrid', 'population': 47.66},
 'germany': {'capital': 'berlin', 'population': 23.66},
 'nigeria': {'capital': 'lagos', 'population': 34.66},
 'usa': {'capital': 'washington', 'population': 97.66}}
```

In [155…
```
print(europe['spain']['capital'])
```

```
madrid
```

In [156…
```
europe['italy'] = {'capital': 'rome', 'population': '34.55'}
europe
```

Out[156]:
```
{'spain': {'capital': 'madrid', 'population': 47.66},
 'germany': {'capital': 'berlin', 'population': 23.66},
 'nigeria': {'capital': 'lagos', 'population': 34.66},
 'usa': {'capital': 'washington', 'population': 97.66},
 'italy': {'capital': 'rome', 'population': '34.55'}}
```

56.

(go to top)

57.

(go to top)

58.

(go to top)

59.

(go to top)

60.

(go to top)

---

61. functions: `definition`

(go to top)

In [3]:
```
def shout(word):
    """ Print string with three exclamation marks"""
    print(word + '!!!')

# Call Shout
shout('Python')
```

```
Python!!!
```

---

62. functions: `multiple arguments`

(go to top)

In [4]:
```
def shout_3(word, word2):
    # Prints string with three exclamation marks
    print(word + word2 +  '!!!')

shout_3('Python', 'Rules')
```

```
PythonRules!!!
```

---

63. functions: `python builtins`

(go to top)

In [6]:
```
import builtins

# Run this if you want to see
#dir(builtins)
```

In [ ]:
```

```

---

64. functions: `return functions`

(go to top)

In [7]:
```
def shout_2(word):
    # Returns string with three exclamation marks
    return word + '!!!'

yell = shout_2('Python')

print(yell)
print(yell, shout_2('Rules'))
```

```
Python!!!
Python!!! Rules!!!
```

Return Multiple Values

(go to top)

In [8]:
```
def shout_all(word1, word2):
    shout1 = word1 + '!!!'
    shout2 = word2 + '!!!'

    return shout1, shout2
```

In [9]:
```
yell1, yell2 = shout_all('Python', 'Rules')
```

In [10]:
```
print(yell1, yell2)
```

```
Python!!! Rules!!!
```

In [11]:
```
shout_all('Python', 'Rules')
```

Out[11]:
```
('Python!!!', 'Rules!!!')
```

---

65. functions: `arguments`

(go to top)

Single default Argument

In [12]:
```
# Define shout_echo

def shout_echo(word1, echo = 1):
    """Concatenate copies of word1 and three
     exclamation marks at the end of the string."""

    # Concatenate echo-copies of word1 using *: echo_word
    echo_word = word1 * echo

    # Concatenate '!!!' to echo_word: shout_word
    shout_word = echo_word + '!!!'

    # Return shout_word
    return shout_word
```

In [13]:
```
# Call shout_echo() with "Hey": no_echo
no_echo = shout_echo("Hey")

# Print no_echo and with_echo
print(no_echo)
```

```
Hey!!!
```

In [14]:
```
# Call shout_echo() with "Hey" and echo=5: with_echo
with_echo = shout_echo("Hey", 5)

# Print with_echo
print(with_echo)
```

```
HeyHeyHeyHeyHey!!!
```

Multiple default Arguments

(go to top)

In [1]:
```
# Define shout_echo
def shout_echo(word1, echo = 1, intense = False):
    """Concatenate copies of word1 and three exclamation marks at the end of

    # Concatenate echo copies of word1 using *: echo_word
    echo_word = word1 * echo

    # Make echo_word uppercase if intense is True
    if intense is True:
        # Make uppercase and concatenate '!!!': echo_word_new
        echo_word_new = echo_word.upper() + '!!!'
    else:
        # Concatenate '!!!' to echo_word: echo_word_new
        echo_word_new = echo_word + '!!!'

    # Return echo_word_new
    return echo_word_new
```

In [3]:
```
# Call shout_echo() with "Hey", echo=5 and intense=True: with_big_echo
with_big_echo = shout_echo("Hey", 5, True)

# Call shout_echo() with "Hey" and intense=True: big_no_echo
big_no_echo = shout_echo("Hey", intense = True)

# Call shout_echo() with "Hey" and intense=True: big_no_echo
just_echo = shout_echo("Hey", 2)
```

In [4]:
```
# Print values
print(with_big_echo)
print(big_no_echo)
print(just_echo)
```

```
HEYHEYHEYHEYHEY!!!
HEY!!!
HeyHey!!!
```

---

66. functions: `args - Variable Length (Positional) Arguments`

(go to top)

In [30]:
```python
def find_type(*args):
    return type(args)

find_type("alpha", 'beta')
```

Out[30]:
```
tuple
```

In [29]:
```python
# Define gibberish
def gibberish(*args):
    """Concatenate strings in *args together."""

    # Initialize an empty string: hodgepodge
    hodgepodge = ""

    # Concatenate the strings in args
    for word in args:
        hodgepodge += word

    # Return hodgepodge
    return hodgepodge
```

In [30]:
```python
# Call gibberish() with one string: one_word
one_word = gibberish('luke')

# Call gibberish() with five strings: many_words
many_words = gibberish("luke", "leia", "han", "obi", "darth")

# Print one_word and many_words
print(one_word)
print(many_words)
```

```
luke
lukeleiahanobidarth
```

---

67. functions: **kwargs – Variable Length Keyword Arguments

(go to top)

In [23]:
```
def find_type(**y):
    return type(y)

find_type(a = "alpha", b = 2)
```

Out[23]:
```
dict
```

In [41]:
```
def find_type(**y):
    for key, value in y.items():
        print(key + ": " , value)
        print(type(value))

find_type(a = "alpha", b = "2", c = 2)
```

```
a:  alpha
<class 'str'>
b:  2
<class 'str'>
c:  2
<class 'int'>
```

In [32]:
```
# Define report_status
def report_status(**kwargs):
    """Print out the status of a movie character."""

    print("\nBEGIN REPORT\n")

    # Iterate over the key-value pairs of kwargs
    for key, value in kwargs.items():
        # Print out the keys and values, separated by a colon ':'
        print(key + ": " + value)

    print("\nEND REPORT")
```

In [33]:
```
# First call to report_status()
report_status(name='luke', affiliation='jedi', status='missing' )

# Second call to report_status()
report_status(name='anakin' , affiliation='sith lord' , status='deceased' )
```

```
BEGIN REPORT

name: luke
affiliation: jedi
status: missing

END REPORT

BEGIN REPORT

name: anakin
affiliation: sith lord
status: deceased

END REPORT
```

---

68. function: `scope`

(go to top)

`testing scope`

In [19]:
```python
#global scope

new_val = 10
```

In [21]:
```python
def square():
    new_val = 5 ** 2
    print(new_val, end=" ||  ")

square()
print(new_val)

# new_val unchanged in the global scope by the function square()
# new_val is accessible, global functions are accesible everywhere but cannc
# without global keyword
```

```
25 ||  10
```

In [21]:
```
def square():
    new_val = 5 ** 2
    print(new_val, end=" ||  ")

square()
print(new_val)

# new_val unchanged in the global scope by the function square()
# new_val is accesible, global functions are accesible everywhere but cannc
# without global keyword
```

```
25 ||  10
```

---

global keyword

- Access & change/affect object in the global scope inside a function

In [ ]:
```
new_val = 10
```

In [41]:
```
def square():

    global new_val

    new_val = new_val ** 2
    print(new_val, end=" ||  ")

square()
print(new_val)

""" new_val IS ACCESSIBLE AND CHANGED in the global scope by the function sq
```

```
100 ||  100
```

Out[41]:
```
' new_val IS ACCESSIBLE AND CHANGED in the global scope by the function squa
re '
```

In [44]:
```
# Create a string: team
team = "teen titans"
```

In [46]:
```
# Define change_team()
def change_team():
    """Change the value of the global variable team."""

    # Use team in global scope
    global team

    # Change the value of team in global: team
    team = "justice league"

    # Print team
    print(team, end="  ||   ")
```

In [47]:
```
# Call change_team()
change_team()

# Print team
print(team)

""" VALUE OF team CHANGES AFTER FUNCTION IS CALLED """
```

Out[47]:
```
justice league  ||   justice league
' VALUE OF team CHANGES AFTER FUNCTION IS CALLED '
```

---

nonlocal keyword

- Acesss and affect an object in an outer function of nested functions

In [22]:
```
def outer():
    """Print n"""
    n = 1

    def inner():
        nonlocal n
        n = 4
        print(n)

    inner()
    print(n)
```

In [49]:
```
outer()
```

```
4
4
```

---

69. functions: `nested functions`

(go to top)

In [23]:
```
# finds the k-root of n
def anyroot(n, k):
    """ Finds the k root of n """
    def root(n):
        return n ** (1/k)
    return root(n)
```

In [24]:
```
print(anyroot(4,2))
```

```
2.0
```

`returns`

In [26]:
```
# Define echo
def echo(n):
    """Returns inner function"""

    def inner_echo(word):
        """Concatenate copies or word"""
        return word * n

    return inner_echo
```

In [27]:
```
echo(2)('test')
```

Out[27]:
```
'testtest'
```

In [28]:
```
twice = echo(2) # repeats the word twice
thrice = echo(3) # repeats the word thrice

print(twice('hey you!'),  "||", thrice('hey there!'))
```

```
hey you!hey you! || hey there!hey there!hey there!
```

In [11]:
```
# Define echo
def echo(n,word):
    """Returns inner function"""

    def inner_echo(n, word):
        """Concatenate copies or word"""
        return word * n

    return inner_echo(n, word)
```

In [21]:
```
print(echo(2, 'Python'))
```

```
PythonPython
```

In [22]:
```
print(echo(3, 'Python'))
```

```
PythonPythonPython
```

```
------------------------------------------------------------------------
----------------------------------------
```

In [17]:
```
def raise_to(x, n):
    """Return x ^ n"""

    def inner(x):
        """ Raise x to the power of n"""
        raised = x ** n
        return raised

    return inner(x)
```

Out[17]:
```
8
```

In [23]:
```
raise_to(2,3)
```

Out[23]:
```
8
```

69b. lambda functions: `passing mutable parameters`

(go to top)

definition

- Parameters are always passed by value. However, if the actual parameter is a variable whose value is a mutable object (like a list or graphics object), then changes to the state of the object will be visible to the calling program.
- The list is passed as a parameter and the change is visible

(go to top)

In [10]:
```
def interest(balances, rate):
    for i in range(len(balances)):
        balances[i] = balances[i] * (1 + rate)
    print(balances)
```

In [11]:
```
def test():
    amounts = [1000,2000,3000,4000]
    rate = 0.05
    interest(amounts,rate)
    print(amounts)
```

In [12]:
```
test()
```

```
[1050.0, 2100.0, 3150.0, 4200.0]
[1050.0, 2100.0, 3150.0, 4200.0]
```

In [17]:
```
def interest(balance, rate):
    balance = balance * (1 + rate)
    print(balance)
```

In [18]:
```
def test():
    amounts = 1000
    rate = 0.05
    interest(amounts,rate)
    print(amounts)
```

In [19]:
```
test()
```

```
1050.0
1000
```

---

70. lambda functions: `definition`

- `lambda input: output`

(go to top)

In [2]:
```
raise_to_power = lambda x, y : x ** y
print(raise_to_power(2,4))
```

```
16
```

In [1]:
```
# Define echo_word as a lambda function
echo_word = (lambda word1, echo: word1 * echo)
echo_word('hey', 5)
```

Out[1]:
```
'heyheyheyheyhey'
```

In [4]:
```
f = lambda a,b: a if (a > b) else b
print(f(5,6))
print(f(9,6))
```

```
6
9
```

---

71. lambda functions: `map( )`

- Takes a function and a sequence such as a list and applies the function over all elemets of the sequence
- map(function, sequence)

(go to top)

In [11]:
```
arr = map(int, input().split())


a = list(arr)
a
```

```
56 78
```

Out[11]:
```
[56, 78]
```

In [12]:
```
numbers = [48, 6, 9, 21, 1]

square_all = map(lambda num: num ** 2, numbers)

print(square_all)
print(list(square_all))
```

```
<map object at 0x7fa0016f5910>
[2304, 36, 81, 441, 1]
```

In [3]:
```
spells = ["protego", "accio", "expecto patronum", "legilimens"]

# Use map() to apply a lambda function over spells: shout_spells
shout_spells = map(lambda word: word + '!!!', spells)

# Print the result
print(list(shout_spells))
```

```
['protego!!!', 'accio!!!', 'expecto patronum!!!', 'legilimens!!!']
```

In [15]:
```
def fahrenheit(T):
    return ((float(9)/5)*T + 32)
def celsius(T):
    return (float(5)/9)*(T-32)
temp = (36.5, 37, 37.5,39)

F = map(fahrenheit, temp)

print(list(F))
```

```
[97.7, 98.60000000000001, 99.5, 102.2]
```

In [4]:
```
fellowship = ['frodo', 'samwise', 'merry', 'pippin', 'aragorn', 'boromir', '

# Use filter() to apply a lambda function over fellowship: result
result_2 = map(lambda member: len(member) > 6 , fellowship)

# Convert result to a list: result_list
result_list = list(result_2)

# Print result_list
print(result_list)
```

```
[False, True, False, False, True, True, True, False, True]
```

---

72. lambda functions: `filter()`

- The function `filter()` offers a way to filter out elements from a list that don't satisfy certain criteria.
- `filter(function, sequence)`

(go to top)

In [18]:
```
fellowship = ['frodo', 'samwise', 'merry', 'pippin', 'aragorn', 'boromir', '

# Use filter() to apply a lambda function over fellowship: result
result = filter(lambda member: len(member) > 6 , fellowship)

# Convert result to a list: result_list
result_list = list(result)

# Print result_list
print(result_list)
```

```
['samwise', 'aragorn', 'boromir', 'legolas', 'gandalf']
```

---

73. lambda functions: `reduce()`

(go to top)

`definition`

- The `reduce()` function is useful for performing some computation on a list
- Note that it returns the final cumulative not step-by-step result. i.e. it runs through whole sequence before giving an answer.
- It always takes 2 lambda parameters and, unlike `map()` and `filter()`, returns a single value as a result.

To use reduce(), you must import it from the functools module.

- The function `reduce(func, seq)` continually applies the function func() to the sequence seq. It returns a single value.
- If `seq = [ s1, s2, s3, ... , sn ]`, calling `reduce(func, seq)` works like this:
  - At first the first two elements of seq will be applied to func, i.e. `func(s1,s2)` The list on which `reduce()` works looks now like this: `[ func(s1, s2), s3, ... , sn ]`
  - In the next step func will be applied on the previous result and the third element of the list, i.e. `func(func(s1, s2),s3)`
  - The list looks like this now: `[ func(func(s1, s2),s3), ... , sn ]`
  - it will continue like this until just one element is left and return this element as the result of `reduce()`

(go to top)

In [7]:
```
# In this exercise, you will use reduce() and a lambda function that concate

# Import reduce from functools
from functools import reduce

# Create a list of strings: stark
stark = ['A', 'sansa', 'arya', 'brandon', 'rickon']

# Use reduce() to apply a lambda function over stark: result
result = reduce(lambda child, child2: child, stark)
print(result)

result = reduce(lambda child, child2: child2, stark)
print(result)
```

```
A
rickon
```

In [11]:
```
result1 = reduce(lambda child, child2: child * 2, stark)
print(result1)
print(len(result1))
```

```
AAAAAAAAAAAAAAAA
16
```

In [12]:
```
result2 = reduce(lambda child, child2: child + child2, stark)
print(result2)
```

```
Asansaaryabrandonrickon
```

```
------------------------------------------------------------------------
----------------------------------------
```

In [15]:
```
print(reduce(lambda x,y: x+y, [47,11,42,13]))
print(sum([47,11,42,13]))
```

```
113
113
```

In [16]:
```
f = lambda a,b: a if (a > b) else b
print(reduce(f, [47,11,42,102,13]))
print(max([47,11,42,102,13]))
```

```
102
102
```

In [17]:
```
print(reduce(lambda x, y: x+y, range(1,101)))
print(sum(range(1,101)))
```

```
5050
5050
```

---

74. loops: `if-if statement`

(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!
```

---

75. loops: `if-else statements`

(go to top)

In [15]:
```python
room = 'bed'
area = 14.0

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

```
looking around elsewhere
```

---

76. loops: `if-elif-else statements`

(go to top)

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

In [16]:
```python
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 [17]:
```python
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(root1, rc

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
```

---

77. loops: `for loops`

(go to top)

`strings as range`

In [25]:
```
message = 'Man'

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

```
M
a
n
```

`range as range`

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

```
0
1
2
```

In [28]:
```
# range 0 to 10, step of 2

for i in range(0,11,2):
    print(i)
```

```
0
2
4
6
8
10
```

nested list as range

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

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
```

dictionary as range

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
```

numpy array as range

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
```

dataframe as range

(go to top)

In [5]:
```
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 [6]:
```
# 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 [7]:
```
# 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
```

---

78. loops: `enumerate`

- Recall that `enumerate()` returns an enumerate object that `produces a sequence of tuples`,
- 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 [8]:
```
avengers = ["hawkeye", "iron man", "thor", "quicksilver"]
```

In [9]:
```
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
```

---

79. loops: `zip()`

- Turn iterables to tuples

(go to top)

In [10]:
```
avengers = ["hawkeye", "iron man", "thor", "quicksilver"]
names = ['barton', 'start', 'odinson', 'maximoff']

z = zip(avengers, names)
print(z)
display(list(z))
```

```
<zip object at 0x7fe6e942dc00>
[('hawkeye', 'barton'),
 ('iron man', 'start'),
 ('thor', 'odinson'),
 ('quicksilver', 'maximoff')]
```

`Splat Operator`

In [13]:
```
mutants = ['charles xavier', 'bobby drake', 'kurt wagner', 'max eisenhardt',
powers =['telepathy', 'thermokinesis', 'teleportation', 'magnetokinesis', 'i

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

```
('charles xavier', 'telepathy') ('bobby drake', 'thermokinesis') ('kurt wagn
er', '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 [16]:
```
# 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
```

---

80. loops: `iter()`

(go to top)

In [21]:
```
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

In [22]:
```
superhero =  iter(flash)

print(*superhero)
```

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

In [23]:
```
word = 'data'
it = iter(word)
print(* it, end="")
```

```
d a t a
```

In [25]:
```
nums = [1, 2, 3, 4, 5]
print (* iter(nums))
```

```
1 2 3 4 5
```

In [26]:
```
nums = [1, 2, 3, 4, 5]
print (* iter(nums), sep='')
```

```
12345
```

---

81. loops: 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
```

---

82. useful code: `simultaneous assignment`

(go to top)

In [1]:
```
x = 2
y = 4

x = y
y = x

print(x,y)
```

```
4 4
```

In [2]:
```
x = 2
y = 4

x,y = y,x

print(x,y)
```

```
4 2
```

In [4]:
```
lst = [1, 2, 3]
lst[0] = lst [2]
lst
# here the value of list[0] is replaced
```

Out[4]:
```
[3, 2, 3]
```

In [3]:
```
lst = [1, 2, 3]
lst[0], lst[2] = lst [2], lst[0]
lst
# here the values are swapped
```

Out[3]:
```
[3, 2, 1]
```

---

83. useful code: `common imports`

(go to top)

In [7]:
```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns
import statsmodels as sm
import datetime as dt
from random import *
import string
import random
import re #regex
from math import * # no need to use math.sqrt() now we can just use sqrt()
```

---

84. useful code: `name = main`

(go to top)

```
def main():
    some function

if__name__ == '__main__':
    #run main
    main()


OR

if__name__ == '__main__':

    some algo
```

84b. useful code: `* unpacking`

(go to top)

In [34]:
```
text = ' hi 24 35 36'

words, *nums = text.split()

print(words)
print(nums)
```

```
hi
['24', '35', '36']
```

84c. useful code: `any()`

(go to top)

In [38]:
```
lst = [1, 2, 3, 4, False]
# is there any item in lst that is True
any(lst)
```

Out[38]:
```
True
```

84d. useful code: `str.isalpha(), str.isdigit(), str.isalnum(), str.islower(), str.isupper()`

(go to top)

In [45]:
```
str.isalpha('9')
```

Out[45]:
```
False
```

In [39]:
```
s = 'yyy'

for method in [str.isalnum, str.isalpha, str.isdigit, str.islower, str.isupp
    print(any(method(c) for c in s))
```

```
True
True
False
True
False
```

84e. useful code: `textwrap`

(go to top)

In [46]:
```
import textwrap

result = textwrap.wrap('randomstring', 2)
print(result)
```

```
['ra', 'nd', 'om', 'st', 'ri', 'ng']
```

In [49]:
```
# without textwrap

str, max_width = input(), int(input())
result = [str[i:i+max_width] for i in range(0, len(str), max_width)]
print('\n'.join(result))
```

```
randomstring
2
ra
nd
om
st
ri
ng
```

In [48]:
```
result = textwrap.wrap('randomstring', 2)
print('-'.join(result))
```

```
ra-nd-om-st-ri-ng
```

84f. useful code: `zip`

(go to top)

In [1]:
```
seq1 = ['foo', 'bar', 'baz']
seq2 = ['one', 'two']
zipped = zip(seq1, seq2)
list(zipped)
```

Out[1]:
```
[('foo', 'one'), ('bar', 'two')]
```

In [8]:
```
seq1 = ['foo', 'bar', 'baz']
seq2 = ['one', 'two', 'three']
zipped = zip(seq1, seq2)
list(zipped)
```

Out[8]:
```
[('foo', 'one'), ('bar', 'two'), ('baz', 'three')]
```

In [9]:
```
pitchers = [('Nolan', 'Ryan'), ('Roger', 'Clemens'),
            ('Curt', 'Schilling')]
first_names, last_names = zip(*pitchers)

print(first_names)
print(last_names)

#print(list(zip(*seq1)))
```

```
('Nolan', 'Roger', 'Curt')
('Ryan', 'Clemens', 'Schilling')
```

84g. useful code: `set`

- A set is an unordered collection of unique elements. You can think of them like dicts, but keys only, no values.
- A set can be created in two ways:
  - via the `set()` function or
  - via a set literal with curly braces `{}`:

(go to top)

In [12]:
```
set([2, 2, 2, 1, 3, 3])
```

Out[12]:
```
{1, 2, 3}
```

In [11]:
```
{2, 2, 2, 1, 3, 4, 4}
```

Out[11]:
```
{1, 2, 3, 4}
```

In [18]:
```
a = {1, 2, 3, 5}
b = {3, 4, 5, 6, 7, 8}

print(a.union(b))
print(a|b)
print(a.intersection(b))
print(a & b)
```

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

In [19]:
```
c = a.copy() #copies a
c
```

Out[19]:
```
{1, 2, 3, 5}
```

In [20]:
```
c |= b #a becomes union of c and b
c
```

Out[20]:
```
{1, 2, 3, 4, 5, 6, 7, 8}
```

In [22]:
```
d = a.copy() #copies a
d &= b       #d becomes intersection of d and b
d
```

Out[22]:
```
{3, 5}
```

- Like dicts, `set elements generally must be immutable`.
- `To have list-like elements, you must convert it to a tuple`:

In [47]:
```
my_data = [1, 2, 3, 4]
my_set = {tuple(my_data)}
my_set
```

Out[47]:
```
{(1, 2, 3, 4)}
```

- You can also check if a set is a `subset of (is contained in)` or a `superset of (contains all elements of)` another set:

In [49]:
```
a_set = {1, 2, 3, 4, 5}
```

In [50]:
```
{1, 2, 3}.issubset(a_set)
```

Out[50]:
```
True
```

In [51]:
```
a_set.issuperset({1, 2, 3})
```

Out[51]:
```
True
```

In [52]:
```
{1, 2, 3} == {3, 2, 1}
```

Out[52]:
```
True
```

84h. useful code: `set comprehensions`

(go to top)

```
set_comp = {expr for value in collection if condition}
```

---

85. print: `newline`

(go to top)

In [10]:
```
print('This program generates a username for you.')
print('This program generates a username for you.')
```

```
This program generates a username for you.
This program generates a username for you.
```

In [9]:
```
print('This program generates a username for you. \n')
print('This program generates a username for you.')
```

```
This program generates a username for you.

This program generates a username for you.
```

---

86. print: `end=''`

(go to top)

In [15]:
```
for i in nums:
    print(i)
```

```
1
2
3
4
5
```

In [13]:
```
for i in nums:
    print(i, end='')
```

```
12345
```

In [16]:
```
for i in nums:
    print(i, end=' ')
```

```
1 2 3 4 5
```

In [17]:
```
for i in nums:
    print(i, end='-')
```

```
1-2-3-4-5-
```

---

87. print: `sep`

(go to top)

In [18]:
```
nums = [1, 2, 3, 4, 5]
print (* iter(nums))
```

```
1 2 3 4 5
```

In [19]:
```
nums = [1, 2, 3, 4, 5]
print (* iter(nums), sep='')
```

```
12345
```

In [22]:
```
nums = [1, 2, 3, 4, 5]
print (* iter(nums), sep='-')
```

```
1-2-3-4-5
```

---

88. Title

(go to top)

---

89. Title

(go to top)

---

90. Title

(go to top)

---

91. Title

(go to top)

---

92. Title

(go to top)

---

93. Title

(go to top)

---

94. Title

(go to top)

---

95. Title

(go to top)

---

96. Title

(go to top)

97. Title

(go to top)

98. Title

(go to top)

99. Title

(go to top)

100. Title

(go to top)

101. Title

(go to top)

102. Title

(go to top)

---

103. Title

(go to top)

. Title

(go to top)

. Title

(go to top)

. Title

(go to top)

. Title

(go to top)

. Title

(go to top)

. Title

(go to top)

. Title

(go to top)

Y. Most Useful Code

(go to top)

Z. Code Challenges

(go to top)

Z. Code Challenges

(go to top)

In [19]:
```
from collections import defaultdict

d = defaultdict(list)
d['python'].append("awesome")
d['something-else'].append("not relevant")
d['python'].append("language")
for i in d.values():
    print(i)

print(d)
```

```
['awesome', 'language']
['not relevant']
defaultdict(<class 'list'>, {'python': ['awesome', 'language'], 'something-e
lse': ['not relevant']})
```

In [ ]:
```
n, m = map(int, input().split())

groupA = [input() for num in range(n)]
groupB = [input() for num in range(m)]

for entry, j in zip(groupB, range(len(groupB))):
    for letter, i in zip(groupA, range(len(groupA))):
        if(i == (len(groupA) - 1)):
            print('', end = '\n')
        if(entry == letter):
            print(groupA.index(letter,i) + 1, end = ' ')
        else:
            print(-1)
```

In [5]:
```
from collections import defaultdict
d = defaultdict(list)
list1=[]

n, m = map(int, input().split())

for i in range(0,n):
    d[input()].append(i+1)

for i in range(0,m):
    list1=list1+[input()]

for i in list1:
    if i in d:
        print (" ".join( map(str,d[i]) ))
    else:
        print (-1)
```

```
5 2
A
A
B
A
B
A
B
1 2 4
3 5
```

In [18]:
```
# https://www.hackerrank.com/challenges/calendar-module/problem

import calendar
# print(calendar.TextCalendar(firstweekday=0).formatyear(2021))
month, day, year = map(int, input().split())
print(day, month, year)
print(calendar.TextCalendar(firstweekday=0).formatmonth(year, month))
day_index = calendar.weekday(year, month, day)
day_name = calendar.day_name[day_index]
print(day_name.upper())
```

```
08 05 2015
5 8 2015
    August 2015
Mo Tu We Th Fr Sa Su
                1  2
 3  4  5  6  7  8  9
10 11 12 13 14 15 16
17 18 19 20 21 22 23
24 25 26 27 28 29 30
31

WEDNESDAY
```

In [ ]:
```
from functools import reduce
```

In [ ]:

```
```

In [21]:

```
#new line
print("There is a double root at")
print("\nThere is a double root at")
```

```
There is a double root at

There is a double root at
```

In [ ]:

```
```

In [25]:

```
nums = [1, 2, 3, 4, 5]
print (* iter(nums))
```

```
1 2 3 4 5
```

In [26]:

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

```
12345
```

In [27]:

```
print?
```

In [ ]:

```
```

In [ ]:

```
```

9. Find Runnerup

- Winner is largest score, runner up is second largest score

(go to top)

In [23]:
```
# assuming inputs are integers only, for float use map(float, input.split())
# collect list of scores
print('Enter scores sperated by space')
arr = map(int, input().split())

# intitialise list of scores to scoreList
scoreList = list(arr)

# set the first score to be the bigges number
# make the runner up
winner = scoreList[0]
runnerUp = winner


# if the next number is greater than the winner make the next number the win
# former winner

# if the next number is less than the current winner but bigger than the run

# if the runner up and the winning number are the same, and the next number
# mae

for i in scoreList:

    if i > winner:
        runnerUp = winner;
        winner = i
        #print('that ran')
    elif i > runnerUp and i < winner:
        runnerUp = i
        #print('this ran')
    elif runnerUp == winner and i < winner:
        runnerUp = i


print('\n', winner, runnerUp)
```

```
Enter scores sperated by space
2 3 5 4 1 2 5 8 1 82 3 5 4 7

 82 8
```

In [ ]: