1. A robot has been given a list of movement instructions. Each instruction is either left, right, up or down, followed by a distance to move. The robot starts at [0, 0]. You want to calculate where the robot will end up and return its final position as a list.

To illustrate, if the robot is given the following instructions: ["right 10", "up 50", "left 30", "down 10"]

It will end up 20 left and 40 up from where it started, so we return [-20, 40].

Examples:

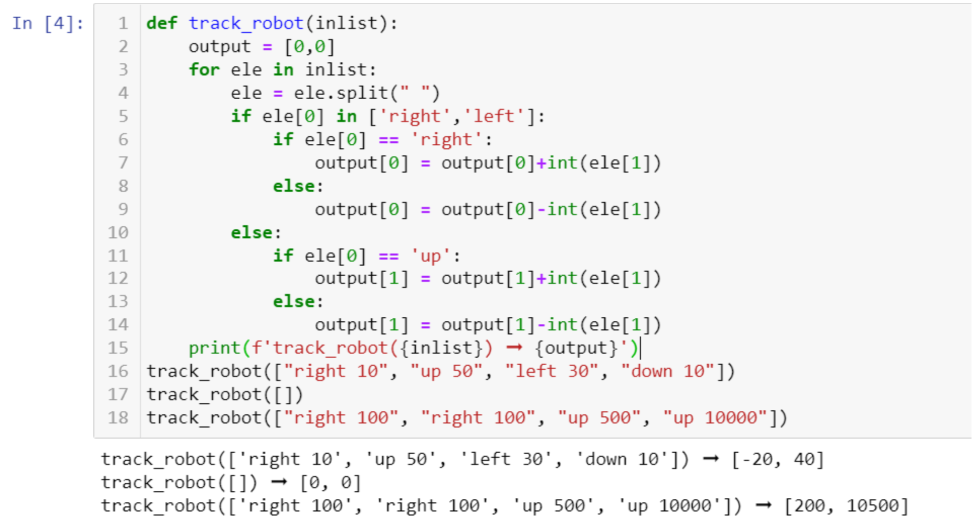
track_robot(["right 10", "up 50", "left 30", "down 10"]) ➞ [-20, 40]

track_robot([]) ➞ [0, 0]

// If there are no instructions, the robot doesn't move.

track_robot(["right 100", "right 100", "up 500", "up 10000"]) ➞ [200, 10500]

Ans:



```
In [4]:
def track_robot(inlist):
    output = [0,0]
    for ele in inlist:
        ele = ele.split(" ")
        if ele[0] in ['right','left']:
            if ele[0] == 'right':
                output[0] = output[0]+int(ele[1])
            else:
                output[0] = output[0]-int(ele[1])
        else:
            if ele[0] == 'up':
                output[1] = output[1]+int(ele[1])
            else:
                output[1] = output[1]-int(ele[1])
    print(f'track_robot({inlist}) ➞ {output}')
track_robot(["right 10", "up 50", "left 30", "down 10"])
track_robot([])
track_robot(["right 100", "right 100", "up 500", "up 10000"])
```

```
track_robot(['right 10', 'up 50', 'left 30', 'down 10']) ➞ [-20, 40]
track_robot([]) ➞ [0, 0]
track_robot(['right 100', 'right 100', 'up 500', 'up 10000']) ➞ [200, 10500]
```

2. Write a function that will return the longest word in a sentence. In cases where more than one word is found, return the first one.

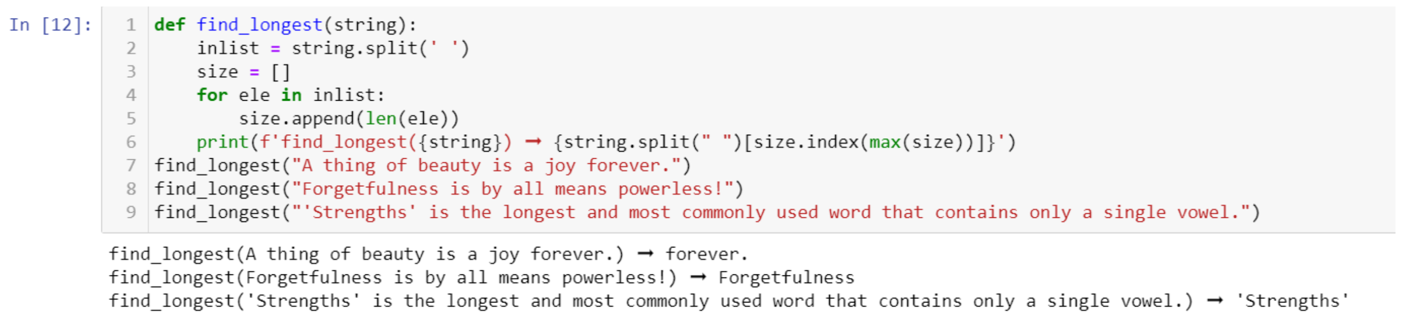
Examples:

find_longest("A thing of beauty is a joy forever.") ➞ "forever"

find_longest("Forgetfulness is by all means powerless!") ➞ "forgetfulness"

find_longest("\"Strengths\" is the longest and most commonly used word that contains only a single vowel.") ➞ "strengths"

Ans:



```
In [12]:
def find_longest(string):
    inlist = string.split(' ')
    size = []
    for ele in inlist:
        size.append(len(ele))
    print(f'find_longest({string}) ➞ {string.split(" ")[size.index(max(size))]}')
find_longest("A thing of beauty is a joy forever.")
find_longest("Forgetfulness is by all means powerless!")
find_longest("'Strengths' is the longest and most commonly used word that contains only a single vowel.")
```

```
find_longest(A thing of beauty is a joy forever.) ➞ forever.
find_longest(Forgetfulness is by all means powerless!) ➞ Forgetfulness
find_longest('Strengths' is the longest and most commonly used word that contains only a single vowel.) ➞ 'Strengths'
```

3. Create a function to check if a candidate is qualified in an imaginary coding interview of an imaginary tech startup.

The criteria for a candidate to be qualified in the coding interview is:

The candidate should have complete all the questions.
The maximum time given to complete the interview is 120 minutes.
The maximum time given for very easy questions is 5 minutes each.
The maximum time given for easy questions is 10 minutes each.
The maximum time given for medium questions is 15 minutes each.
The maximum time given for hard questions is 20 minutes each.
If all the above conditions are satisfied, return "qualified", else return "disqualified".

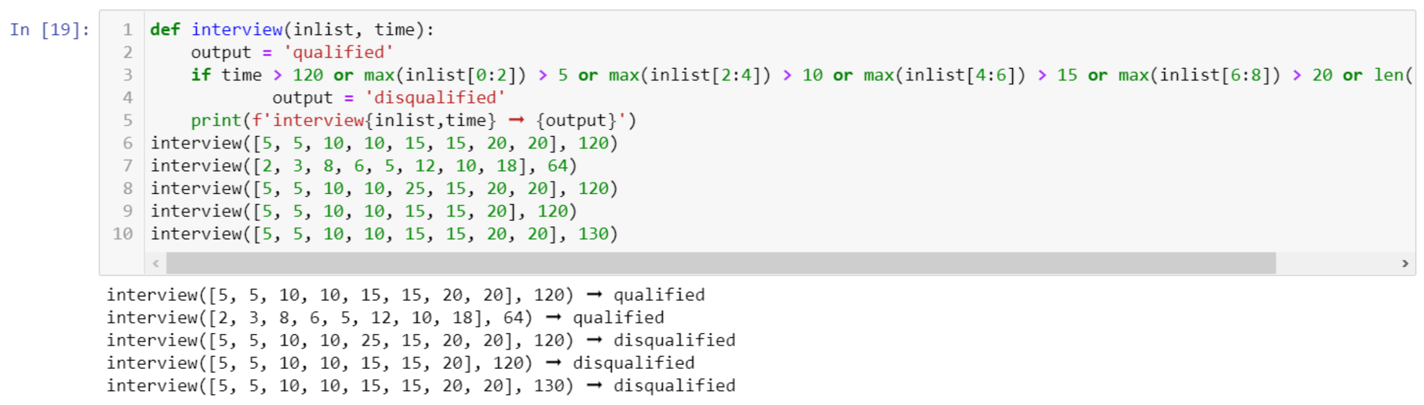
You will be given a list of time taken by a candidate to solve a particular question and the total time taken by the candidate to complete the interview.

Given a list , in a true condition will always be in the format [very easy, very easy, easy, easy, medium, medium, hard, hard].

The maximum time to complete the interview includes a buffer time of 20 minutes.

Examples:
interview([5, 5, 10, 10, 15, 15, 20, 20], 120) ➞ "qualified"
interview([2, 3, 8, 6, 5, 12, 10, 18], 64) ➞ "qualified"
interview([5, 5, 10, 10, 25, 15, 20, 20], 120) ➞ "disqualified"
# Exceeded the time limit for a medium question.
interview([5, 5, 10, 10, 15, 15, 20], 120) ➞ "disqualified"
# Did not complete all the questions.
interview([5, 5, 10, 10, 15, 15, 20, 20], 130) ➞ "disqualified"
# Solved all the questions in their respected time limits but exceeded the total time limit of the interview.

Ans:



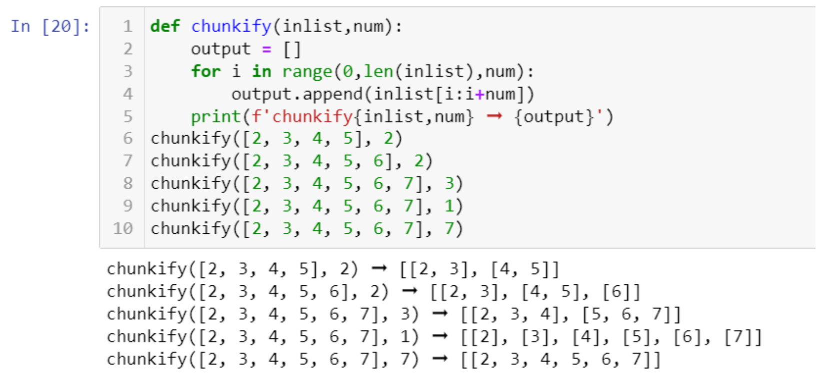
```
In [19]:
def interview(inlist, time):
    output = 'qualified'
    if time > 120 or max(inlist[0:2]) > 5 or max(inlist[2:4]) > 10 or max(inlist[4:6]) > 15 or max(inlist[6:8]) > 20 or len(
        output = 'disqualified'
    print(f'interview{inlist,time} ➞ {output}')
interview([5, 5, 10, 10, 15, 15, 20, 20], 120)
interview([2, 3, 8, 6, 5, 12, 10, 18], 64)
interview([5, 5, 10, 10, 25, 15, 20, 20], 120)
interview([5, 5, 10, 10, 15, 15, 20], 120)
interview([5, 5, 10, 10, 15, 15, 20, 20], 130)
```

```
interview([5, 5, 10, 10, 15, 15, 20, 20], 120) ➞ qualified
interview([2, 3, 8, 6, 5, 12, 10, 18], 64) ➞ qualified
interview([5, 5, 10, 10, 25, 15, 20, 20], 120) ➞ disqualified
interview([5, 5, 10, 10, 15, 15, 20], 120) ➞ disqualified
interview([5, 5, 10, 10, 15, 15, 20, 20], 130) ➞ disqualified
```

4. Write a function that divides a list into chunks of size n, where n is the length of each chunk.

Examples:
chunkify([2, 3, 4, 5], 2) ➞ [[2, 3], [4, 5]]
chunkify([2, 3, 4, 5, 6], 2) ➞ [[2, 3], [4, 5], [6]]
chunkify([2, 3, 4, 5, 6, 7], 3) ➞ [[2, 3, 4], [5, 6, 7]]
chunkify([2, 3, 4, 5, 6, 7], 1) ➞ [[2], [3], [4], [5], [6], [7]]
chunkify([2, 3, 4, 5, 6, 7], 7) ➞ [[2, 3, 4, 5, 6, 7]]

Ans:



```
In [20]:
def chunkify(inlist,num):
    output = []
    for i in range(0,len(inlist),num):
        output.append(inlist[i:i+num])
    print(f'chunkify{inlist,num} ➞ {output}')
chunkify([2, 3, 4, 5], 2)
chunkify([2, 3, 4, 5, 6], 2)
chunkify([2, 3, 4, 5, 6, 7], 3)
chunkify([2, 3, 4, 5, 6, 7], 1)
chunkify([2, 3, 4, 5, 6, 7], 7)
```

```
chunkify([2, 3, 4, 5], 2) ➞ [[2, 3], [4, 5]]
chunkify([2, 3, 4, 5, 6], 2) ➞ [[2, 3], [4, 5], [6]]
chunkify([2, 3, 4, 5, 6, 7], 3) ➞ [[2, 3, 4], [5, 6, 7]]
chunkify([2, 3, 4, 5, 6, 7], 1) ➞ [[2], [3], [4], [5], [6], [7]]
chunkify([2, 3, 4, 5, 6, 7], 7) ➞ [[2, 3, 4, 5, 6, 7]]
```

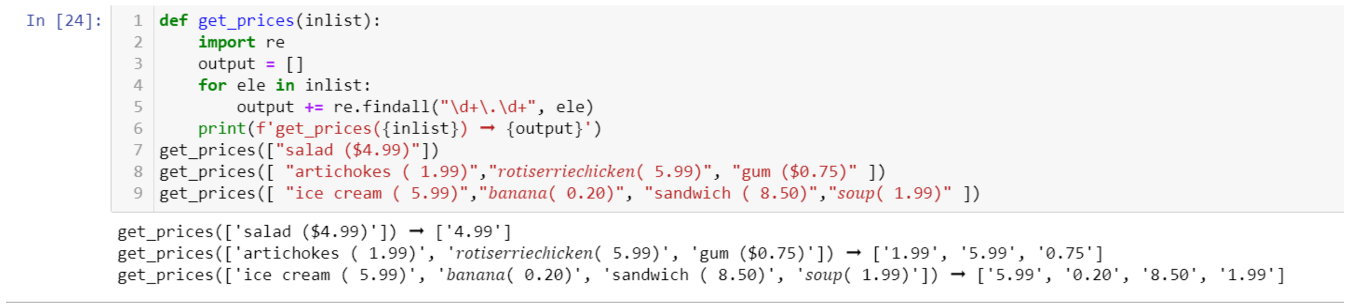
5. You are given a list of strings consisting of grocery items, with prices in parentheses. Return a list of prices in float format.

Examples:
get_prices(["salad ($4.99)"]) ➞ [4.99]

get_prices([
  "artichokes ($1.99)",
  "rotiserrie chicken ($5.99)",
  "gum ($0.75)"
]) ➞ [1.99, 5.99, 0.75]

get_prices([
  "ice cream ($5.99)",
  "banana ($0.20)",
  "sandwich ($8.50)",
  "soup ($1.99)"
]) ➞ [5.99, 0.2, 8.50, 1.99]

Ans:



In [24]:
```python
def get_prices(inlist):
    import re
    output = []
    for ele in inlist:
        output += re.findall("\d+\.\d+", ele)
    print(f'get_prices({inlist}) ➞ {output}')
get_prices(["salad ($4.99)"])
get_prices([ "artichokes ( 1.99)","rotiserriechicken( 5.99)", "gum ($0.75)" ])
get_prices([ "ice cream ( 5.99)","banana( 0.20)", "sandwich ( 8.50)","soup( 1.99)" ])
```

```
get_prices(['salad ($4.99)']) ➞ ['4.99']
get_prices(['artichokes ( 1.99)', 'rotiserriechicken( 5.99)', 'gum ($0.75)']) ➞ ['1.99', '5.99', '0.75']
get_prices(['ice cream ( 5.99)', 'banana( 0.20)', 'sandwich ( 8.50)', 'soup( 1.99)']) ➞ ['5.99', '0.20', '8.50', '1.99']
```