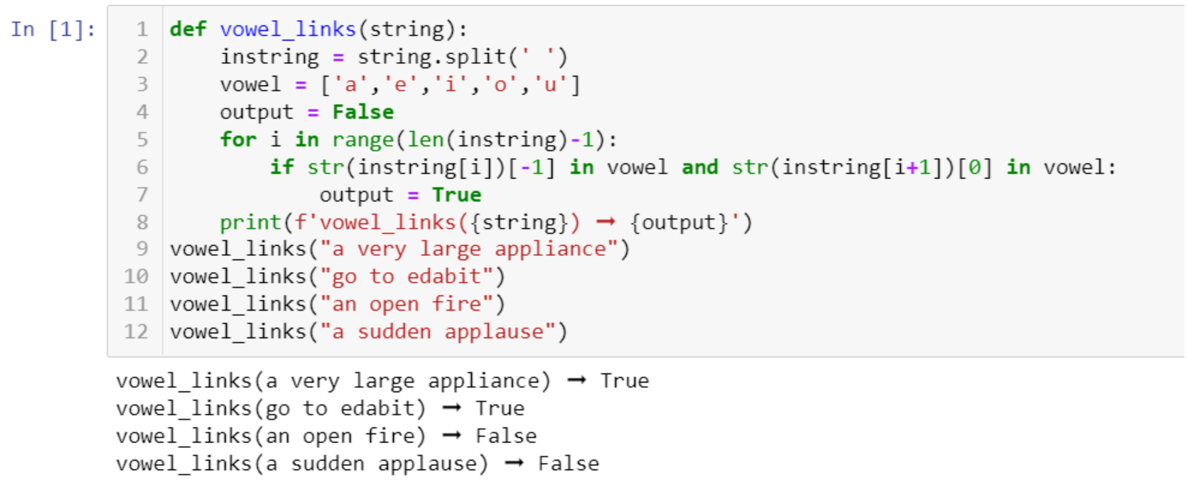
1. Given a sentence as txt, return True if any two adjacent words have this property: One word ends with a vowel, while the word immediately after begins with a vowel (a e i o u).

Examples:
vowel_links("a very large appliance") ➞ True
vowel_links("go to edabit") ➞ True
vowel_links("an open fire") ➞ False
vowel_links("a sudden applause") ➞ False

Ans:



```
def vowel_links(string):
    instring = string.split(' ')
    vowel = ['a','e','i','o','u']
    output = False
    for i in range(len(instring)-1):
        if str(instring[i])[-1] in vowel and str(instring[i+1])[0] in vowel:
            output = True
    print(f'vowel_links({string}) ➞ {output}')
vowel_links("a very large appliance")
vowel_links("go to edabit")
vowel_links("an open fire")
vowel_links("a sudden applause")
```

```
vowel_links(a very large appliance) ➞ True
vowel_links(go to edabit) ➞ True
vowel_links(an open fire) ➞ False
vowel_links(a sudden applause) ➞ False
```

2. You are given three inputs: a string, one letter, and a second letter. Write a function that returns True if every instance of the first letter occurs before every instance of the second letter.

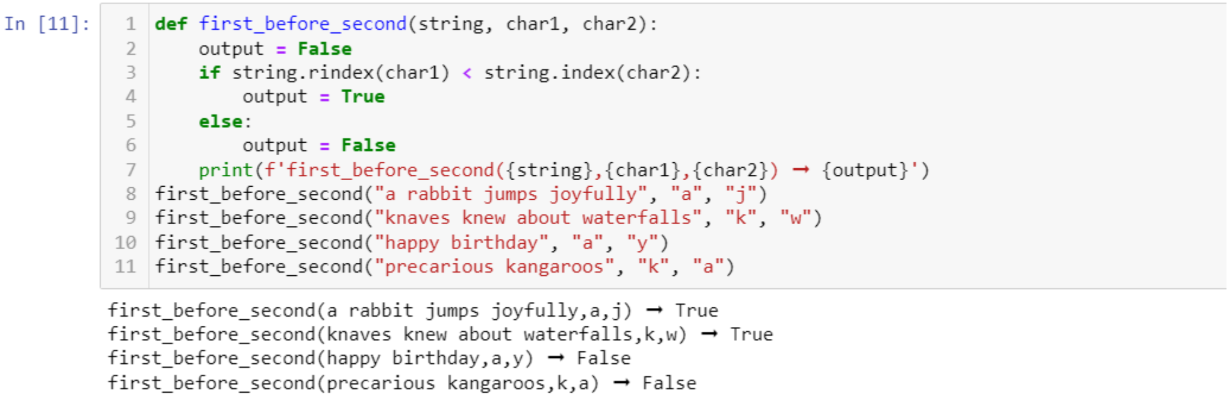
Examples:
first_before_second("a rabbit jumps joyfully", "a", "j") ➞ True
# Every instance of "a" occurs before every instance of "j".

first_before_second("knaves knew about waterfalls", "k", "w") ➞ True

first_before_second("happy birthday", "a", "y") ➞ False
# The "a" in "birthday" occurs after the "y" in "happy".

first_before_second("precarious kangaroos", "k", "a") ➞ False

Ans:



In [11]:

```python
def first_before_second(string, char1, char2):
    output = False
    if string.rindex(char1) < string.index(char2):
        output = True
    else:
        output = False
    print(f'first_before_second({string},{char1},{char2}) ➞ {output}')
first_before_second("a rabbit jumps joyfully", "a", "j")
first_before_second("knaves knew about waterfalls", "k", "w")
first_before_second("happy birthday", "a", "y")
first_before_second("precarious kangaroos", "k", "a")
```

```
first_before_second(a rabbit jumps joyfully,a,j) ➞ True
first_before_second(knaves knew about waterfalls,k,w) ➞ True
first_before_second(happy birthday,a,y) ➞ False
first_before_second(precarious kangaroos,k,a) ➞ False
```

3. Create a function that returns the characters from a list or string r on odd or even positions, depending on the specifier s. The specifier will be "odd" for items on odd positions (1, 3, 5, ...) and "even" for items on even positions (2, 4, 6, ...).

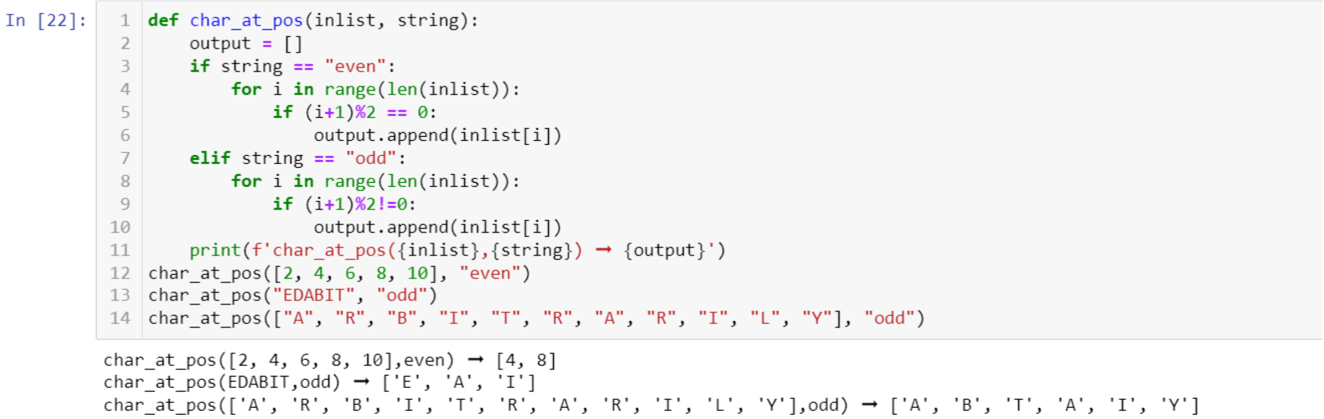
Examples:

char_at_pos([2, 4, 6, 8, 10], "even") ➞ [4, 8]
# 4 & 8 occupy the 2nd & 4th positions

char_at_pos("EDABIT", "odd") ➞ "EAI"
# "E", "A" and "I" occupy the 1st, 3rd and 5th positions

char_at_pos(["A", "R", "B", "I", "T", "R", "A", "R", "I", "L", "Y"], "odd") ➞ ["A", "B", "T", "A", "I", "Y"]

Ans:



In [22]:

```python
def char_at_pos(inlist, string):
    output = []
    if string == "even":
        for i in range(len(inlist)):
            if (i+1)%2 == 0:
                output.append(inlist[i])
    elif string == "odd":
        for i in range(len(inlist)):
            if (i+1)%2!=0:
                output.append(inlist[i])
    print(f'char_at_pos({inlist},{string}) ➞ {output}')
char_at_pos([2, 4, 6, 8, 10], "even")
char_at_pos("EDABIT", "odd")
char_at_pos(["A", "R", "B", "I", "T", "R", "A", "R", "I", "L", "Y"], "odd")
```

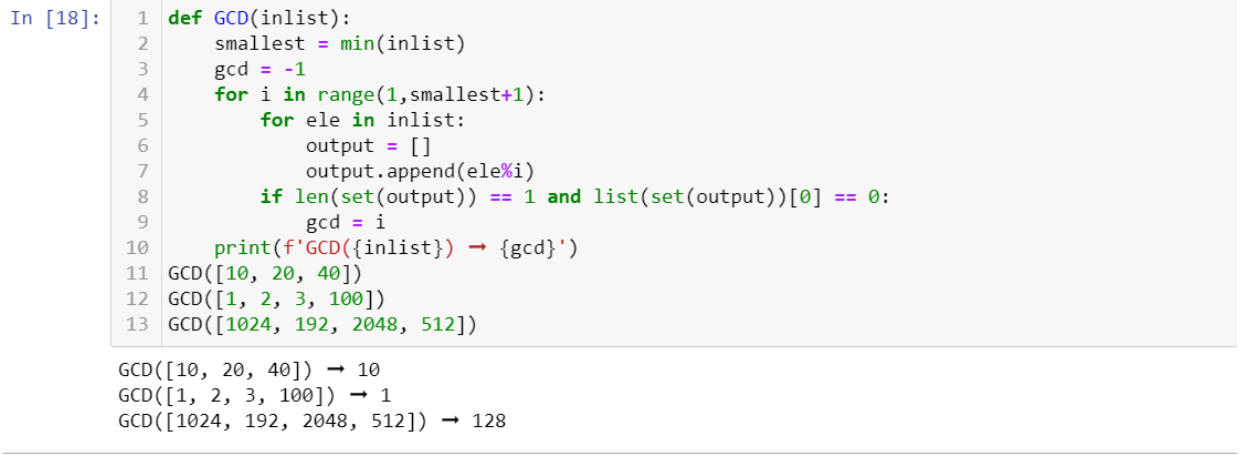
```
char_at_pos([2, 4, 6, 8, 10],even) ➞ [4, 8]
char_at_pos(EDABIT,odd) ➞ ['E', 'A', 'I']
char_at_pos(['A', 'R', 'B', 'I', 'T', 'R', 'A', 'R', 'I', 'L', 'Y'],odd) ➞ ['A', 'B', 'T', 'A', 'I', 'Y']
```

4. Write a function that returns the greatest common divisor of all list elements. If the greatest common divisor is 1, return 1.

Examples:

GCD([10, 20, 40]) ➞ 10
GCD([1, 2, 3, 100]) ➞ 1
GCD([1024, 192, 2048, 512]) ➞ 64

Ans:



```python
def GCD(inlist):
    smallest = min(inlist)
    gcd = -1
    for i in range(1,smallest+1):
        for ele in inlist:
            output = []
            output.append(ele%i)
        if len(set(output)) == 1 and list(set(output))[0] == 0:
            gcd = i
    print(f'GCD({inlist}) ➞ {gcd}')
GCD([10, 20, 40])
GCD([1, 2, 3, 100])
GCD([1024, 192, 2048, 512])
```

```
GCD([10, 20, 40]) ➞ 10
GCD([1, 2, 3, 100]) ➞ 1
GCD([1024, 192, 2048, 512]) ➞ 128
```

5. A number/string is a palindrome if the digits/characters are the same when read both forward and backward. Examples include "racecar" and 12321. Given a positive number n, check if n or the binary representation of n is palindromic. Return the following:

"Decimal only." if only n is a palindrome.
"Binary only." if only the binary representation of n is a palindrome.
"Decimal and binary." if both are palindromes.
"Neither!" if neither are palindromes.
Examples:

palindrome_type(1306031) ➞ "Decimal only."
# decimal = 1306031
# binary = "100111110110110101111"

palindrome_type(427787) ➞ "Binary only."
# decimal = 427787
# binary = "1101000011100001011"

palindrome_type(313) ➞ "Decimal and binary."
# decimal = 313
# binary = 100111001

palindrome_type(934) ➞ "Neither!"
# decimal = 934
# binary = "1110100110"

Ans:

In [29]:

```
def palindrome_type(n):
    N = str(n)
    b = str(bin(n))[2:]
    if N == N[: :-1] and b != b[: :-1]:
        print(f'palindrome_type({n}) ➞ "Decimal only." decimal = {n} binary = {b}')
    elif N != N[: :-1] and b == b[: :-1]:
        print(f'palindrome_type({n}) ➞ "Binary only." decimal = {n} binary = {b}')
    elif N == N[: :-1] and b == b[: :-1]:
        print(f'palindrome_type({n}) ➞ "Decimal and binary." decimal = {n} binary = {b}')
    else:
        print(f'palindrome_type({n}) ➞ "Neither!" decimal = {n} binary = {b}')
palindrome_type(1306031)
palindrome_type(427787)
palindrome_type(313)
palindrome_type(934)
```

```
palindrome_type(1306031) ➞ "Decimal only." decimal = 1306031 binary = 100111110110110101111
palindrome_type(427787) ➞ "Binary only." decimal = 427787 binary = 1101000011100001011
palindrome_type(313) ➞ "Decimal and binary." decimal = 313 binary = 100111001
palindrome_type(934) ➞ "Neither!" decimal = 934 binary = 1110100110
```