In [1]:
```
"""
Q1 - Write a Python program to find those numbers which are divisible by 7
and multiples of 5, between 1500 and 2700 (both included).
"""
# Create an empty list to store numbers that meet the given conditions
nl = []

# Iterate through numbers from 1500 to 2700 (inclusive)
for x in range(1500, 2701):
    # Check if the number is divisible by 7 and 5 without any remainder
    if (x % 7 == 0) and (x % 5 == 0):
        # If the conditions are met, convert the number to a string and append it to the list
        nl.append(str(x))

# Join the numbers in the list with a comma and print the result
print(','.join(nl))
```

```
1505,1540,1575,1610,1645,1680,1715,1750,1785,1820,1855,1890,1925,1960,1995,2030,2065,2100,2135,2170,2205,2240,2275,2310,23
45,2380,2415,2450,2485,2520,2555,2590,2625,2660,2695
```

In [2]:
```
"""
Q2 - Write a Python function that takes a sequence of numbers and determines
whether all the numbers are different from each other.
"""
# Define a function named test_distinct that takes a list 'data' as a parameter.
def test_distinct(data):
    # Check if the length of the list is equal to the length of the set created from the list.
    if len(data) == len(set(data)):
        # If the lengths are equal, it means all elements in the list are distinct.
        return True
    else:
        # If the lengths are not equal, there are duplicate elements in the list.
        return False

# Call the test_distinct function with a list [1, 5, 7, 9] and print the result.
print(test_distinct([1, 5, 7, 9]))

# Call the test_distinct function with a list [2, 4, 5, 5, 7, 9] and print the result.
print(test_distinct([2, 4, 5, 5, 7, 9]))
```

```
True
False
```

In [7]:
```python
from itertools import permutations

# Define the vowels
vowels = ['a', 'e', 'i', 'o', 'I']

# Generate all permutations of the vowels
perms = permutations(vowels)
print(type(perms))

# Iterate through the permutations and join the characters to form strings
for perm in perms:
  print(perm)
  print(''.join(perm))
```

```
<class 'itertools.permutations'>
('a', 'e', 'i', 'o', 'I')
aeioI
('a', 'e', 'i', 'I', 'o')
aeiIo
('a', 'e', 'o', 'i', 'I')
aeoiI
('a', 'e', 'o', 'I', 'i')
aeoIi
('a', 'e', 'I', 'i', 'o')
aeIio
('a', 'e', 'I', 'o', 'i')
aeIoi
('a', 'i', 'e', 'o', 'I')
aieoI
('a', 'i', 'e', 'I', 'o')
aieIo
('a', 'i', 'o', 'e', 'I')
aioeI
('a', 'i', 'o', 'I', 'e')
aioIe
('a', 'i', 'I', 'e', 'o')
aiIeo
('a', 'i', 'I', 'o', 'e')
aiIoe
('a', 'o', 'e', 'i', 'I')
aoeiI
('a', 'o', 'e', 'I', 'i')
aoeIi
('a', 'o', 'i', 'e', 'I')
aoieI
('a', 'o', 'i', 'I', 'e')
aoiIe
('a', 'o', 'I', 'e', 'i')
aoIei
('a', 'o', 'I', 'i', 'e')
aoIie
('a', 'I', 'e', 'i', 'o')
aIeio
('a', 'I', 'e', 'o', 'i')
aIeoi
('a', 'I', 'i', 'e', 'o')
aIieo
('a', 'I', 'i', 'o', 'e')
aIioe
```

```
('a', 'I', 'o', 'e', 'i')
aIoei
('a', 'I', 'o', 'i', 'e')
aIoie
('e', 'a', 'i', 'o', 'I')
eaioI
('e', 'a', 'i', 'I', 'o')
eaiIo
('e', 'a', 'o', 'i', 'I')
eaoiI
('e', 'a', 'o', 'I', 'i')
eaoIi
('e', 'a', 'I', 'i', 'o')
eaIio
('e', 'a', 'I', 'o', 'i')
eaIoi
('e', 'i', 'a', 'o', 'I')
eiaoI
('e', 'i', 'a', 'I', 'o')
eiaIo
('e', 'i', 'o', 'a', 'I')
eioaI
('e', 'i', 'o', 'I', 'a')
eioIa
('e', 'i', 'I', 'a', 'o')
eiIao
('e', 'i', 'I', 'o', 'a')
eiIoa
('e', 'o', 'a', 'i', 'I')
eoaiI
('e', 'o', 'a', 'I', 'i')
eoaIi
('e', 'o', 'i', 'a', 'I')
eoiaI
('e', 'o', 'i', 'I', 'a')
eoiIa
('e', 'o', 'I', 'a', 'i')
eoIai
('e', 'o', 'I', 'i', 'a')
eoIia
('e', 'I', 'a', 'i', 'o')
eIaio
('e', 'I', 'a', 'o', 'i')
eIaoi
('e', 'I', 'i', 'a', 'o')
```

```
eIiao
('e', 'I', 'i', 'o', 'a')
eIioa
('e', 'I', 'o', 'a', 'i')
eIoai
('e', 'I', 'o', 'i', 'a')
eIoia
('i', 'a', 'e', 'o', 'I')
iaeoI
('i', 'a', 'e', 'I', 'o')
iaeIo
('i', 'a', 'o', 'e', 'I')
iaoeI
('i', 'a', 'o', 'I', 'e')
iaoIe
('i', 'a', 'I', 'e', 'o')
iaIeo
('i', 'a', 'I', 'o', 'e')
iaIoe
('i', 'e', 'a', 'o', 'I')
ieaoI
('i', 'e', 'a', 'I', 'o')
ieaIo
('i', 'e', 'o', 'a', 'I')
ieoaI
('i', 'e', 'o', 'I', 'a')
ieoIa
('i', 'e', 'I', 'a', 'o')
ieIao
('i', 'e', 'I', 'o', 'a')
ieIoa
('i', 'o', 'a', 'e', 'I')
ioaeI
('i', 'o', 'a', 'I', 'e')
ioaIe
('i', 'o', 'e', 'a', 'I')
ioeaI
('i', 'o', 'e', 'I', 'a')
ioeIa
('i', 'o', 'I', 'a', 'e')
ioIae
('i', 'o', 'I', 'e', 'a')
ioIea
('i', 'I', 'a', 'e', 'o')
iIaeo
```

```
('i', 'I', 'a', 'o', 'e')
iIaoe
('i', 'I', 'e', 'a', 'o')
iIeao
('i', 'I', 'e', 'o', 'a')
iIeoa
('i', 'I', 'o', 'a', 'e')
iIoae
('i', 'I', 'o', 'e', 'a')
iIoea
('o', 'a', 'e', 'i', 'I')
oaeiI
('o', 'a', 'e', 'I', 'i')
oaeIi
('o', 'a', 'i', 'e', 'I')
oaieI
('o', 'a', 'i', 'I', 'e')
oaiIe
('o', 'a', 'I', 'e', 'i')
oaIei
('o', 'a', 'I', 'i', 'e')
oaIie
('o', 'e', 'a', 'i', 'I')
oeaiI
('o', 'e', 'a', 'I', 'i')
oeaIi
('o', 'e', 'i', 'a', 'I')
oeiaI
('o', 'e', 'i', 'I', 'a')
oeiIa
('o', 'e', 'I', 'a', 'i')
oeIai
('o', 'e', 'I', 'i', 'a')
oeIia
('o', 'i', 'a', 'e', 'I')
oiaeI
('o', 'i', 'a', 'I', 'e')
oiaIe
('o', 'i', 'e', 'a', 'I')
oieaI
('o', 'i', 'e', 'I', 'a')
oieIa
('o', 'i', 'I', 'a', 'e')
oiIae
('o', 'i', 'I', 'e', 'a')
```

```
oiIea
('o', 'I', 'a', 'e', 'i')
oIaei
('o', 'I', 'a', 'i', 'e')
oIaie
('o', 'I', 'e', 'a', 'i')
oIeai
('o', 'I', 'e', 'i', 'a')
oIeia
('o', 'I', 'i', 'a', 'e')
oIiae
('o', 'I', 'i', 'e', 'a')
oIiea
('I', 'a', 'e', 'i', 'o')
Iaeio
('I', 'a', 'e', 'o', 'i')
Iaeoi
('I', 'a', 'i', 'e', 'o')
Iaieo
('I', 'a', 'i', 'o', 'e')
Iaioe
('I', 'a', 'o', 'e', 'i')
Iaoei
('I', 'a', 'o', 'i', 'e')
Iaoie
('I', 'e', 'a', 'i', 'o')
Ieaio
('I', 'e', 'a', 'o', 'i')
Ieaoi
('I', 'e', 'i', 'a', 'o')
Ieiao
('I', 'e', 'i', 'o', 'a')
Ieioa
('I', 'e', 'o', 'a', 'i')
Ieoai
('I', 'e', 'o', 'i', 'a')
Ieoia
('I', 'i', 'a', 'e', 'o')
Iiaeo
('I', 'i', 'a', 'o', 'e')
Iiaoe
('I', 'i', 'e', 'a', 'o')
Iieao
('I', 'i', 'e', 'o', 'a')
Iieoa
```

```
('I', 'i', 'o', 'a', 'e')
Iioae
('I', 'i', 'o', 'e', 'a')
Iioea
('I', 'o', 'a', 'e', 'i')
Ioaei
('I', 'o', 'a', 'i', 'e')
Ioaie
('I', 'o', 'e', 'a', 'i')
Ioeai
('I', 'o', 'e', 'i', 'a')
Ioeia
('I', 'o', 'i', 'a', 'e')
Ioiae
('I', 'o', 'i', 'e', 'a')
Ioiea
```

In [12]:

```python
#How to create a series from a list, numpy array and dict?
import numpy as np
import pandas as pd

mylist = list('abcedfghijklmnopqrstuvwxyz') #list
myarr = np.arange(26) # numpy array
print(myarr)
print(type(myarr))
mydict=dict(zip(mylist,myarr)) #Using zip function we are creating a dictionary using list and np array
print(mydict)
ser1 = pd.Series(mylist)
print(ser1)
ser2 = pd.Series(myarr)
print(ser2)
ser3 = pd.Series(mydict)
print(ser3)
```

```
[ 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]
<class 'numpy.ndarray'>
{'a': 0, 'b': 1, 'c': 2, 'e': 3, 'd': 4, 'f': 5, 'g': 6, 'h': 7, 'i': 8, 'j': 9, 'k': 10, 'l': 11, 'm': 12, 'n': 13, 'o':
14, 'p': 15, 'q': 16, 'r': 17, 's': 18, 't': 19, 'u': 20, 'v': 21, 'w': 22, 'x': 23, 'y': 24, 'z': 25}
0    a
1    b
2    c
3    e
4    d
5    f
6    g
7    h
8    i
9    j
10   k
11   l
12   m
13   n
14   o
15   p
16   q
17   r
18   s
19   t
20   u
21   v
22   w
23   x
24   y
25   z
dtype: object
0     0
1     1
2     2
3     3
4     4
5     5
6     6
7     7
8     8
9     9
10   10
11   11
12   12
```

```
13    13
14    14
15    15
16    16
17    17
18    18
19    19
20    20
21    21
22    22
23    23
24    24
25    25
dtype: int32
a     0
b     1
c     2
e     3
d     4
f     5
g     6
h     7
i     8
j     9
k    10
l    11
m    12
n    13
o    14
p    15
q    16
r    17
s    18
t    19
u    20
v    21
w    22
x    23
y    24
z    25
dtype: int32
```

In [24]:
```
#How to convert the index of a series into a column of a dataframe?
mylist=list('abcedfghijklmnopqrstuvwxyz')
ser1=pd.Series(mylist)
```

```
df = ser1.to_frame().reset_index()
print(df.head())
```

```
   index  0
0      0  a
1      1  b
2      2  c
3      3  e
4      4  d
```

In [28]:

```
# How to combine many series to form a dataframe?
mylist = list('abcedfghijklmnopqrstuvwxyz') #list
myarr = np.arange(26) # numpy array
mydict=dict(zip(mylist,myarr))
ser1 = pd.Series(mylist)
ser2 = pd.Series(myarr)

#1st Approach
df = pd.concat([ser1,ser2],axis=1)
print(df)

#2nd Approach
df2 = pd.DataFrame({'col1':ser1,'col2':ser2})
print(df2)
```

	0	1
0	a	0
1	b	1
2	c	2
3	e	3
4	d	4
5	f	5
6	g	6
7	h	7
8	i	8
9	j	9
10	k	10
11	l	11
12	m	12
13	n	13
14	o	14
15	p	15
16	q	16
17	r	17
18	s	18
19	t	19
20	u	20
21	v	21
22	w	22
23	x	23
24	y	24
25	z	25

	col1	col2
0	a	0
1	b	1
2	c	2
3	e	3
4	d	4
5	f	5
6	g	6
7	h	7
8	i	8
9	j	9
10	k	10
11	l	11
12	m	12
13	n	13
14	o	14
15	p	15
16	q	16

```
17    r    17
18    s    18
19    t    19
20    u    20
21    v    21
22    w    22
23    x    23
24    y    24
25    z    25
```

In [32]:
```
#How to assign name to the series' index?
mylist = list('abcedfghijklmnopqrstuvwxyz') #list
ser1 = pd.Series(mylist)
ser1.name='alphabets'
ser1.head()
```

Out[32]:
```
0    a
1    b
2    c
3    e
4    d
Name: alphabets, dtype: object
```

In [39]:
```
# How to get the items of series A not present in series B?
ser1 = pd.Series([1, 2, 3, 4, 5])
ser2 = pd.Series([4, 5, 6, 7, 8])
ser1[~ser1.isin(ser2)]
```

Out[39]:
```
0    1
1    2
2    3
dtype: int64
```

In [48]:
```
#How to get the minimum, 25th percentile, median, 75th, and max of a numeric series?
ser = pd.Series(np.random.normal(10, 5, 25))
print(np.min(ser))
print(np.max(ser))
print(np.median(ser))
print(np.mean(ser))
print(np.percentile(ser,q=[0,25,50,75,100]))
#print(np.std())
print(dir(ser))
print(ser.std())
```

```
2.4002721685099786
16.522655178218898
8.656145903268987
8.714305100051948
[ 2.40027217  5.74754928  8.6561459  10.93427329 16.52265518]
['T', '_AXIS_LEN', '_AXIS_ORDERS', '_AXIS_TO_AXIS_NUMBER', '_HANDLED_TYPES', '__abs__', '__add__', '__and__', '__annotatio
ns__', '__array__', '__array_priority__', '__array_ufunc__', '__bool__', '__class__', '__contains__', '__copy__', '__deepc
opy__', '__delattr__', '__delitem__', '__dict__', '__dir__', '__divmod__', '__doc__', '__eq__', '__finalize__', '__float_
_', '__floordiv__', '__format__', '__ge__', '__getattr__', '__getattribute__', '__getitem__', '__getstate__', '__gt__', '_
_hash__', '__iadd__', '__iand__', '__ifloordiv__', '__imod__', '__imul__', '__init__', '__init_subclass__', '__int__', '__
invert__', '__ior__', '__ipow__', '__isub__', '__iter__', '__itruediv__', '__ixor__', '__le__', '__len__', '__lt__', '__ma
tmul__', '__mod__', '__module__', '__mul__', '__ne__', '__neg__', '__new__', '__nonzero__', '__or__', '__pos__', '__pow_
_', '__radd__', '__rand__', '__rdivmod__', '__reduce__', '__reduce_ex__', '__repr__', '__rfloordiv__', '__rmatmul__', '__r
mod__', '__rmul__', '__ror__', '__round__', '__rpow__', '__rsub__', '__rtruediv__', '__rxor__', '__setattr__', '__setitem_
_', '__setstate__', '__sizeof__', '__str__', '__sub__', '__subclasshook__', '__truediv__', '__weakref__', '__xor__', '_acc
essors', '_accum_func', '_add_numeric_operations', '_agg_examples_doc', '_agg_see_also_doc', '_align_frame', '_align_serie
s', '_append', '_arith_method', '_as_manager', '_attrs', '_binop', '_can_hold_na', '_check_inplace_and_allows_duplicate_la
bels', '_check_inplace_setting', '_check_is_chained_assignment_possible', '_check_label_or_level_ambiguity', '_check_setit
em_copy', '_clear_item_cache', '_clip_with_one_bound', '_clip_with_scalar', '_cmp_method', '_consolidate', '_consolidate_i
nplace', '_construct_axes_dict', '_construct_result', '_constructor', '_constructor_expanddim', '_convert_dtypes', '_dat
a', '_dir_additions', '_dir_deletions', '_drop_axis', '_drop_labels_or_levels', '_duplicated', '_find_valid_index', '_flag
s', '_get_axis', '_get_axis_name', '_get_axis_number', '_get_axis_resolvers', '_get_block_manager_axis', '_get_bool_data',
'_get_cacher', '_get_cleaned_column_resolvers', '_get_index_resolvers', '_get_label_or_level_values', '_get_numeric_data',
'_get_value', '_get_values', '_get_values_tuple', '_get_with', '_gotitem', '_hidden_attrs', '_indexed_same', '_info_axis',
'_info_axis_name', '_info_axis_number', '_init_dict', '_init_mgr', '_inplace_method', '_internal_names', '_internal_names_
set', '_is_cached', '_is_copy', '_is_label_or_level_reference', '_is_label_reference', '_is_level_reference', '_is_mixed_t
ype', '_is_view', '_item_cache', '_ixs', '_logical_func', '_logical_method', '_map_values', '_maybe_update_cacher', '_memo
ry_usage', '_metadata', '_mgr', '_min_count_stat_function', '_name', '_needs_reindex_multi', '_protect_consolidate', '_red
uce', '_references', '_reindex_axes', '_reindex_indexer', '_reindex_multi', '_reindex_with_indexers', '_rename', '_replace
_single', '_repr_data_resource_', '_repr_latex_', '_reset_cache', '_reset_cacher', '_set_as_cached', '_set_axis', '_set_ax
is_name', '_set_axis_nocheck', '_set_is_copy', '_set_labels', '_set_name', '_set_value', '_set_values', '_set_with', '_set
_with_engine', '_slice', '_stat_axis', '_stat_axis_name', '_stat_axis_number', '_stat_function', '_stat_function_ddof', '_
take', '_take_with_is_copy', '_to_latex_via_styler', '_typ', '_update_inplace', '_validate_dtype', '_values', '_where', 'a
bs', 'add', 'add_prefix', 'add_suffix', 'agg', 'aggregate', 'align', 'all', 'any', 'apply', 'argmax', 'argmin', 'argsort',
'array', 'asfreq', 'asof', 'astype', 'at', 'at_time', 'attrs', 'autocorr', 'axes', 'backfill', 'between', 'between_time',
'bfill', 'bool', 'clip', 'combine', 'combine_first', 'compare', 'convert_dtypes', 'copy', 'corr', 'count', 'cov', 'cumma
x', 'cummin', 'cumprod', 'cumsum', 'describe', 'diff', 'div', 'divide', 'divmod', 'dot', 'drop', 'drop_duplicates', 'dropl
evel', 'dropna', 'dtype', 'dtypes', 'duplicated', 'empty', 'eq', 'equals', 'ewm', 'expanding', 'explode', 'factorize', 'ff
ill', 'fillna', 'filter', 'first', 'first_valid_index', 'flags', 'floordiv', 'ge', 'get', 'groupby', 'gt', 'hasnans', 'hea
d', 'hist', 'iat', 'idxmax', 'idxmin', 'iloc', 'index', 'infer_objects', 'info', 'interpolate', 'is_monotonic_decreasing',
'is_monotonic_increasing', 'is_unique', 'isin', 'isna', 'isnull', 'item', 'items', 'keys', 'kurt', 'kurtosis', 'last', 'la
st_valid_index', 'le', 'loc', 'lt', 'map', 'mask', 'max', 'mean', 'median', 'memory_usage', 'min', 'mod', 'mode', 'mul',
'multiply', 'name', 'nbytes', 'ndim', 'ne', 'nlargest', 'notna', 'notnull', 'nsmallest', 'nunique', 'pad', 'pct_change',
'pipe', 'plot', 'pop', 'pow', 'prod', 'product', 'quantile', 'radd', 'rank', 'ravel', 'rdiv', 'rdivmod', 'reindex', 'reind
ex_like', 'rename', 'rename_axis', 'reorder_levels', 'repeat', 'replace', 'resample', 'reset_index', 'rfloordiv', 'rmod',
```

```
'rmul', 'rolling', 'round', 'rpow', 'rsub', 'rtruediv', 'sample', 'searchsorted', 'sem', 'set_axis', 'set_flags', 'shape',
'shift', 'size', 'skew', 'sort_index', 'sort_values', 'squeeze', 'std', 'sub', 'subtract', 'sum', 'swapaxes', 'swaplevel',
'tail', 'take', 'to_clipboard', 'to_csv', 'to_dict', 'to_excel', 'to_frame', 'to_hdf', 'to_json', 'to_latex', 'to_list',
'to_markdown', 'to_numpy', 'to_period', 'to_pickle', 'to_sql', 'to_string', 'to_timestamp', 'to_xarray', 'transform', 'tra
nspose', 'truediv', 'truncate', 'tz_convert', 'tz_localize', 'unique', 'unstack', 'update', 'value_counts', 'values', 'va
r', 'view', 'where', 'xs']
3.6813612006094796
```

In [57]:

```
#How to keep only top 2 most frequent values as it is and replace everything else as 'Other'?
np.random.RandomState(100)
#np.random.randint(1, 5, [12]) #input data to create a series
ser = pd.Series(np.random.randint(1, 5, [12]))
#print("Top 2 Freq:", ser.value_counts())
ser[~ser.isin(ser.value_counts().index[:2])] = 'Other'
ser
```

Out[57]:

```
0        1
1        4
2        4
3        4
4        1
5        4
6    Other
7    Other
8    Other
9        4
10       1
11   Other
dtype: object
```

In [62]:

```
#How to bin a numeric series to 10 groups of equal size?
'''
Bin the series ser into 10 equal deciles and replace the values with the bin name.
Desired Output
# First 5 items
0    7th
1    9th
2    7th
3    3rd
4    8th
dtype: category
Categories (10, object): [1st < 2nd < 3rd < 4th ... 7th < 8th < 9th < 10th]
'''
ser = pd.Series(np.random.random(20))
#print(ser.head())
```

```
pd.qcut(ser,q=[0, .10, .20, .3, .4, .5, .6, .7, .8, .9, 1],
        labels=['1st', '2nd', '3rd', '4th', '5th', '6th', '7th', '8th', '9th', '10th'])
```

Out[62]:

```
0     8th
1    10th
2     3rd
3     1st
4     8th
5     9th
6     5th
7     9th
8     6th
9     5th
10    4th
11    7th
12    3rd
13    7th
14    6th
15    2nd
16    1st
17    4th
18   10th
19    2nd
dtype: category
Categories (10, object): ['1st' < '2nd' < '3rd' < '4th' ... '7th' < '8th' < '9th' < '10th']
```

In [63]:

```
#How to convert a numpy array to a dataframe of given shape?
ser = pd.Series(np.random.randint(1, 10, 35))
df=pd.DataFrame(ser.values.reshape(7,5))
df
```

Out[63]:

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

In [64]:
```
# Create a DataFrame from a NumPy array with custom column names.
import pandas as pd
import numpy as np

# Create a NumPy array
numpy_array = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])

# Define custom column names
column_names = ['Column1', 'Column2', 'Column3']

# Create a DataFrame with custom column names
df = pd.DataFrame(data=numpy_array, columns=column_names)

# Display the DataFrame
print(df)
```

```
   Column1  Column2  Column3
0        1        2        3
1        4        5        6
2        7        8        9
```

In [67]:
```
#Filtering rows based on a column condition in Pandas DataFrame
# Create a sample DataFrame
data = {'Name': ['Teodosija', 'Sutton', 'Taneli', 'Ravshan', 'Ross', 'Alice', 'Bob', 'Charlie', 'David', 'Emily'],
        'Age': [26, 32, 25, 31, 28, 22, 35, 30, 40, 28],
        'Salary': [50000, 60000, 45000, 70000, 55000, 60000, 70000, 55000, 75000, 65000]}

df = pd.DataFrame(data)

# Filter rows where Age is greater than 30
filtered_rows = df[df['Age'] > 30]

# Display the filtered rows
print(filtered_rows)
df.head(7)
df.tail(7)
```

```
      Name  Age  Salary
1   Sutton   32   60000
3  Ravshan   31   70000
6      Bob   35   70000
8    David   40   75000
```

Out[67]:

	Name	Age	Salary
3	Ravshan	31	70000
4	Ross	28	55000
5	Alice	22	60000
6	Bob	35	70000
7	Charlie	30	55000
8	David	40	75000
9	Emily	28	65000

In [73]:
```
#Change the first character of each word to upper case in each word of ser.
ser = pd.Series(['how', 'to', 'kick', 'ass?'])
#Approach 1
print(ser.map(lambda x: x.title()))
print("******************Approach 2******************")
#Approach 2
print(ser.map(lambda x: x[0].upper() + x[1:]))
print("******************Approach 3******************")
#Approach 3
pd.Series([i.title() for i in ser])
```

```
0    How
1     To
2   Kick
3   Ass?
dtype: object
******************Approach 2******************
0    How
1     To
2   Kick
3   Ass?
dtype: object
******************Approach 3******************
```

Out[73]:
```
0    How
1     To
2   Kick
3   Ass?
dtype: object
```

In [ ]: