Prácticas MD

Francisco Javier Mercader Martínez

Actividad 1. Realizar las siguientes operaciones con Python:

$$\text{a) } (2^4+3)^2 \qquad \text{b) } \frac{2+4^4}{1+\frac{2}{4\cdot 3^3}} \qquad \text{c) } \left(\frac{4+^3}{2}+5^3\right)^6$$

$$\text{d) } 1+2\frac{7}{2^4+5} \qquad \text{e) } (2+3^2+5^3)^{\frac{1}{3}} \qquad \text{f) } \left(1+2^3\frac{5}{2^4+1}\right)^{\frac{1}{2}}$$

[1]:
```
print(f"a) {(2**4+3)**2}")
print(f"b) {(2+4**4)/(1+(2/(4*3**3)))}")
print(f"c) {((4+4**3)/2+5**2)**6}")
print(f"d) {1+2*(7/(2**4+5))}")
print(f"e) {(2+3**2+5**3)**(1/3)}")
print(f"f) {(1+2**3*(5/(2**4+1)))**(1/2)}")
```

```
a) 361
b) 253.30909090909088
c) 42180533641.0
d) 1.6666666666666665
e) 5.14256318131647
f) 1.8311038136792213
```

Actividad 2. Obtener el resto y el cociente de las siguientes divisiones enteras:

a) 45 entre 3 b) 111 entre 67
c) 99 entre 54 d) 103964 entre 78

[2]:
```
print(f"a)\n Cociente:{45//3} \n Resto:{45%3}")
print(f"b)\n Cociente:{111//67} \n Resto:{111%67}")
print(f"c)\n Cociente:{99//54} \n Resto:{99%54}")
print(f"d)\n Cociente:{103964//78} \n Resto:{103964%78}")
```

```
a)
 Cociente:15
 Resto:0
b)
 Cociente:1
 Resto:44
c)
```

```
 Cociente:1
 Resto:45
d)
 Cociente:1332
 Resto:68
```

Actividad 3. Dadas las listas A de los 10 primeros números naturales pares y B de los 5 primeros múltiplos de 3, hacer las siguientes operaciones:

1. Hacer la unión de A y B. LLamar C a esta nueva lista.
2. Eliminar los elementos repetidos de C eliminando el elemento repetido que aparece en primer lugar.
3. Añadir a la lista resultante los números 5 y 7 al final de la lista.
4. Eliminar los elementos repetidos de C eliminando el elemento repetido que aparece en último lugar.
5. Eliminar los elementos repetidos de C eliminando el elemento repetido que aparece en último lugar.
6. Crear una nueva lista D con los elementos pares de C, sin escribir el número en cuestión, sino seleccionándolo de la lista C.

[3]:
```python
# Apartado 1
A=[]
B=[]

for i in range(10):
    A.append(i+1)

for j in range(5):
    B.append((j+1)*3)

C = A + B

print(f"A={A}")
print(f"B={B}")
print(f"C={C}")

# Apartado 2
for n in C:
    while(C.count(n) > 1):
        C.remove(n)
print(C)

# Apartado 3
C.append(5)
C.append(7)
print(C)
```

```
# Apartado 4
lista = [3,4,5]

C = lista + C
print(C)

# Apartado 5
for i in range(len(C) - 1, 0, -1):
    if C[i] in C[:i]:
        C.pop(i)
print(C)

# Apartado 6
D = []
for i in C:
    if i%2==0:
        D.append(i)

print(D)
```

```
A=[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
B=[3, 6, 9, 12, 15]
C=[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 3, 6, 9, 12, 15]
[1, 2, 4, 5, 7, 8, 10, 3, 6, 9, 12, 15]
[1, 2, 4, 5, 7, 8, 10, 3, 6, 9, 12, 15, 5, 7]
[3, 4, 5, 1, 2, 4, 5, 7, 8, 10, 3, 6, 9, 12, 15, 5, 7]
[3, 4, 5, 1, 2, 7, 8, 10, 6, 9, 12, 15]
[4, 2, 8, 10, 6, 12]
```

Actividad 4. Dada la función $f(x) = 3.95x(1-x)$ y $x_0 = 0.5$, obtener los 100 primeros elementos de la recursión

$$x_{n+1} = f(x_n).$$

[4]:
```
import numpy as np

def f(x):
    return (3.95*x*(1-x))

x4 = np.zeros(100)

x4[0] = 0.5
for i in range(1,100):
    x4[i] = f(x4[i-1])

for j in range(len(x4)):
    print(f"x_{j} = {x4[j]}")
```

```
x_0 = 0.5
x_1 = 0.9875
x_2 = 0.04875781249999983
x_3 = 0.183202928469848
x_4 = 0.591076481106183
x_5 = 0.9547350446277946
x_6 = 0.17070335479006285
x_7 = 0.5591766918412491
x_8 = 0.9736675706137671
x_9 = 0.10127417856796533
x_10 = 0.35951999132722934
x_11 = 0.9095482002950283
x_12 = 0.3249675729586585
x_13 = 0.8664864154618691
x_14 = 0.4569664437635456
x_15 = 0.9801850464986934
x_16 = 0.07671816842023815
x_17 = 0.2797883396652044
x_18 = 0.7959519573777409
x_19 = 0.6415291337509211
x_20 = 0.9083795419838698
x_21 = 0.3287432912717266
x_22 = 0.8716510018764594
x_23 = 0.44190835457668476
x_24 = 0.9741701748914467
x_25 = 0.09939244871148847
x_26 = 0.35357867990995934
x_27 = 0.9028151482412049
x_28 = 0.3465728275722937
x_29 = 0.8945174059053136
x_30 = 0.3727062649290811
x_31 = 0.9234954047961941
x_32 = 0.27907398636020553
x_33 = 0.7947072011640559
x_34 = 0.6444332790490923
x_35 = 0.9050991602173518
x_36 = 0.3392839480452255
x_37 = 0.8854728850440775
x_38 = 0.4005720868383919
x_39 = 0.9484506558330944
x_40 = 0.1931234366673267
x_41 = 0.6155157607646546
x_42 = 0.9347916306091042
x_43 = 0.24077713991149136
x_44 = 0.7220738597897574
x_45 = 0.7926986431524128
x_46 = 0.6490936419721093
x_47 = 0.8996957893977955
```

```
x_48 = 0.3564609399538088
x_49 = 0.906116326052171
x_50 = 0.33602464236985274
x_51 = 0.8812927242557581
x_52 = 0.41323264079700855
x_53 = 0.9577621302389092
x_54 = 0.15979263687058687
x_55 = 0.5303228527864999
x_56 = 0.9838680721656087
x_57 = 0.06269317051801239
x_58 = 0.23211281070922665
x_59 = 0.7040339925648631
x_60 = 0.8230620130182635
x_61 = 0.5752421961911199
x_62 = 0.9651375170537312
x_63 = 0.13290600640490524
x_64 = 0.4552058994722979
x_65 = 0.979574279803761
x_66 = 0.07903361509528065
x_67 = 0.2875098459819941
x_68 = 0.8091493410593463
x_69 = 0.6099854054441592
x_70 = 0.9397176818276715
x_71 = 0.22376102313798982
x_72 = 0.6860835092658077
x_73 = 0.8507230639383221
x_74 = 0.5016236630657768
x_75 = 0.987489586687083
x_76 = 0.0487979163430452
x_77 = 0.18334588482930833
x_78 = 0.5914341768145921
x_79 = 0.9544771756754398
x_80 = 0.17162946232079807
x_81 = 0.5615825204378737
x_82 = 0.9725199930472495
x_83 = 0.10556318187397795
x_84 = 0.37295740620114665
x_85 = 0.9237477084753952
x_86 = 0.2782296242693133
x_87 = 0.7932307067706668
x_88 = 0.6478622227967622
x_89 = 0.9011402141249226
x_90 = 0.3518917880166544
x_91 = 0.9008526322952307
x_92 = 0.35280281036883904
x_93 = 0.9019153000905179
x_94 = 0.3494331616349383
x_95 = 0.8979520273797601
```

```
x_96 = 0.3619550264221378
x_97 = 0.9122271618160548
x_98 = 0.3162716298912667
x_99 = 0.854163349767894
```

Actividad 5. Dada la recursión de la actividad 4, obtener los 100 primeros elementos pares, es decir, los de la sucesión $x_0, x_2, x_4, x_6, \ldots$

[5]:
```
import numpy as np

def f(x):
    return (3.95*x*(1-x))

x = np.zeros(200)

x[0] = 0.5
print(f"x_0 = {x[0]}")

for i in range(1,200):
    x[i] = f(x[i-1])
    if i % 2 == 0:
        print(f"x_{i} = {x[i]}")
```

```
x_0 = 0.5
x_2 = 0.04875781249999983
x_4 = 0.591076481106183
x_6 = 0.17070335479006285
x_8 = 0.9736675706137671
x_10 = 0.35951999132722934
x_12 = 0.3249675729586585
x_14 = 0.4569664437635456
x_16 = 0.07671816842023815
x_18 = 0.7959519573777409
x_20 = 0.9083795419838698
x_22 = 0.8716510018764594
x_24 = 0.9741701748914467
x_26 = 0.35357867990995934
x_28 = 0.3465728275722937
x_30 = 0.3727062649290811
x_32 = 0.27907398636020553
x_34 = 0.6444332790490923
x_36 = 0.3392839480452255
x_38 = 0.4005720868383919
x_40 = 0.1931234366673267
x_42 = 0.9347916306091042
x_44 = 0.7220738597897574
x_46 = 0.6490936419721093
x_48 = 0.3564609399538088
x_50 = 0.33602464236985274
```

x_52 = 0.41323264079700855
x_54 = 0.15979263687058687
x_56 = 0.9838680721656087
x_58 = 0.23211281070922665
x_60 = 0.8230620130182635
x_62 = 0.9651375170537312
x_64 = 0.4552058994722979
x_66 = 0.07903361509528065
x_68 = 0.8091493410593463
x_70 = 0.9397176818276715
x_72 = 0.6860835092658077
x_74 = 0.5016236630657768
x_76 = 0.0487979163430452
x_78 = 0.5914341768145921
x_80 = 0.17162946232079807
x_82 = 0.9725199930472495
x_84 = 0.37295740620114665
x_86 = 0.2782296242693133
x_88 = 0.6478622227967622
x_90 = 0.3518917880166544
x_92 = 0.35280281036883904
x_94 = 0.3494331616349383
x_96 = 0.3619550264221378
x_98 = 0.3162716298912667
x_100 = 0.49204487064067837
x_102 = 0.049720270839305
x_104 = 0.5996076384403604
x_106 = 0.19362394377429912
x_108 = 0.933679416980738
x_110 = 0.7298297760941513
x_112 = 0.6803493626260131
x_114 = 0.47835559976717507
x_116 = 0.055871010026073496
x_118 = 0.6515378020282968
x_120 = 0.36559235540186297
x_122 = 0.30346333026396255
x_124 = 0.5444105542351017
x_126 = 0.07852152477740656
x_128 = 0.8062773110444152
x_130 = 0.9334588325364238
x_132 = 0.731351998615936
x_134 = 0.6864277291644301
x_136 = 0.503025992862475
x_138 = 0.048897102036946666
x_140 = 0.5923180078073083
x_142 = 0.1739310704686262
x_144 = 0.969485605681602
x_146 = 0.40763631372603204

```
x_148 = 0.17405053589544103
x_150 = 0.9693210830054336
x_152 = 0.4094815187419775
x_154 = 0.1692648491204359
x_156 = 0.9753652930346579
x_158 = 0.33931313756757653
x_160 = 0.400459233166295
x_162 = 0.19343762952952226
x_164 = 0.9340945839444051
x_166 = 0.7269504199189561
x_168 = 0.6687980636722092
x_170 = 0.4321690151352626
x_172 = 0.11744620476504807
x_174 = 0.9550968333930211
x_176 = 0.555787592747178
x_178 = 0.09550584959890672
x_180 = 0.8879145764780896
x_182 = 0.942371928051122
x_184 = 0.6655637317008795
x_186 = 0.4194436669022315
x_188 = 0.14488091524723745
x_190 = 0.9870534288252106
x_192 = 0.18931941723260337
x_194 = 0.9429195984312224
x_196 = 0.661229916218985
x_198 = 0.4025603833594322
```

Actividad 6. Dada la recursión de la actividad 4, obtener los 100 primeros elementos múltiplos de 4, es decir, los de la sucesión x_0, x_4, x_8, x_{12}

[6]:
```
import numpy as np

def f(x):
    return (3.95*x*(1-x))

x = np.zeros(400)

x[0] = 0.5
print(f"x_0 = {x[0]}")

for i in range(1,400):
    x[i] = f(x[i-1])
    if i % 4 == 0:
        print(f"x_{i} = {x[i]}")
```

```
x_0 = 0.5
x_4 = 0.591076481106183
x_8 = 0.9736675706137671
x_12 = 0.3249675729586585
```

```
x_16 = 0.07671816842023815
x_20 = 0.9083795419838698
x_24 = 0.9741701748914467
x_28 = 0.3465728275722937
x_32 = 0.27907398636020553
x_36 = 0.3392839480452255
x_40 = 0.1931234366673267
x_44 = 0.7220738597897574
x_48 = 0.3564609399538088
x_52 = 0.41323264079700855
x_56 = 0.9838680721656087
x_60 = 0.8230620130182635
x_64 = 0.4552058994722979
x_68 = 0.8091493410593463
x_72 = 0.6860835092658077
x_76 = 0.0487979163430452
x_80 = 0.17162946232079807
x_84 = 0.37295740620114665
x_88 = 0.6478622227967622
x_92 = 0.35280281036883904
x_96 = 0.3619550264221378
x_100 = 0.49204487064067837
x_104 = 0.5996076384403604
x_108 = 0.933679416980738
x_112 = 0.6803493626260131
x_116 = 0.055871010026073496
x_120 = 0.36559235540186297
x_124 = 0.5444105542351017
x_128 = 0.8062773110444152
x_132 = 0.731351998615936
x_136 = 0.503025992862475
x_140 = 0.5923180078073083
x_144 = 0.969485605681602
x_148 = 0.17405053589544103
x_152 = 0.4094815187419775
x_156 = 0.9753652930346579
x_160 = 0.400459233166295
x_164 = 0.9340945839444051
x_168 = 0.6687980636722092
x_172 = 0.11744620476504807
x_176 = 0.555787592747178
x_180 = 0.8879145764780896
x_184 = 0.6655637317008795
x_188 = 0.14488091524723745
x_192 = 0.18931941723260337
x_196 = 0.661229916218985
x_200 = 0.18763634616776279
x_204 = 0.6334801720948651
```

```
x_208 = 0.5576887664660336
x_212 = 0.9002669401320579
x_216 = 0.889658396228557
x_220 = 0.7008861420609539
x_224 = 0.10685923426372336
x_228 = 0.7690558227342181
x_232 = 0.9707878977981979
x_236 = 0.21513151585969176
x_240 = 0.9652302833330871
x_244 = 0.08037386921642775
x_248 = 0.9542731845105783
x_252 = 0.10905860832617482
x_256 = 0.7264508196060054
x_260 = 0.42424487344492495
x_264 = 0.9806839884871316
x_268 = 0.6672195577407398
x_272 = 0.1303299103894285
x_276 = 0.3231593329224524
x_280 = 0.06813402697257925
x_284 = 0.7290107163276519
x_288 = 0.46514116018195406
x_292 = 0.7355399628983461
x_296 = 0.5711425472062732
x_300 = 0.9679115908713581
x_304 = 0.13206647659900914
x_308 = 0.2986186052891005
x_312 = 0.11065598467542272
x_316 = 0.6945602833238884
x_320 = 0.06869093504798097
x_324 = 0.7433170209744371
x_328 = 0.6944477064640189
x_332 = 0.06818935966399586
x_336 = 0.7304462438816353
x_340 = 0.48833169879300115
x_344 = 0.6092880012968456
x_348 = 0.8574012778210138
x_352 = 0.19888274418689014
x_356 = 0.8066416672212617
x_360 = 0.7257221207451932
x_364 = 0.41274593939786913
x_368 = 0.9830492280209023
x_372 = 0.7851067509627749
x_376 = 0.9639622809861216
x_380 = 0.06461979343242154
x_384 = 0.6321731710048271
x_388 = 0.5763105657203039
x_392 = 0.9817946656532536
x_396 = 0.7236780891275244
```

Actividad 7. Dada la función $f(x) = 3.95x(1-x)$ y $x_0 = 0.5$, $x_1 = 0.25$, obtener los 100 primeros elementos de la recursión

$$x_{n+1} = 0.25 \cdot x_{n-1} + 0.75 \cdot f(x_n)$$

[7]:

```
import numpy as np

def f(x):
    return (3.95*x*(1-x))

x7 = np.zeros(100)

x7[0] = 0.5
x7[1] = 0.25

for i in range(2,100):
    x7[i] = 0.25 * x7[i-2] + 0.75*f(x7[i-1])

for j in range(len(x7)):
    print(f"x_{j} = {x7[j]}")
```

```
x_0 = 0.5
x_1 = 0.25
x_2 = 0.6804687500000001
x_3 = 0.7066394271850587
x_4 = 0.7842438733804189
x_5 = 0.6779309148666401
x_6 = 0.8428949648417141
x_7 = 0.5617859938595264
x_8 = 0.9400393706876955
x_9 = 0.3074288544900332
x_10 = 0.8657745411509747
x_11 = 0.4211263318206727
x_12 = 0.9386387582740082
x_13 = 0.27590985068443286
x_14 = 0.8265186193209428
x_15 = 0.49375727670996594
x_16 = 0.9471392014827857
x_17 = 0.27176142762585154
x_18 = 0.8230847443345053
x_19 = 0.49932849154219844
x_20 = 0.946394850222435
x_21 = 0.2751245995566608
x_22 = 0.8274132108466229
x_23 = 0.4918278958773633
x_24 = 0.9472804572274992
x_25 = 0.2709047944945801
x_26 = 0.821959447744622
```

```
x_27 = 0.5012647113722799
x_28 = 0.946110123432647
x_29 = 0.2763614852393863
x_30 = 0.8289855069533245
x_31 = 0.4890796598457889
x_32 = 0.9475180872696707
x_33 = 0.2695878161021899
x_34 = 0.8202260648950639
x_35 = 0.5042331835849462
x_36 = 0.9456284286880968
x_37 = 0.2783761326482333
x_38 = 0.8315225841289121
x_39 = 0.4846188826915552
x_40 = 0.9478047814271218
x_41 = 0.2677121959509991
x_42 = 0.8177267345236808
x_43 = 0.5084878509131706
x_44 = 0.9448432544270399
x_45 = 0.28151110673841884
x_46 = 0.8354137765386983
x_47 = 0.4777144122646974
x_48 = 0.9480071261508275
x_49 = 0.2654490872609761
x_50 = 0.8146474194375809
x_51 = 0.5136908885850094
x_52 = 0.9437315625847882
x_53 = 0.2857382869807117
x_54 = 0.8405551987107909
x_55 = 0.46847521076467463
x_56 = 0.9478196306313151
x_57 = 0.2636368787640699
x_58 = 0.8120723646071705
x_59 = 0.5180188309646669
x_60 = 0.942681231778893
x_61 = 0.28957843906968084
x_62 = 0.8451240042805803
x_63 = 0.460154521462834
x_64 = 0.9472025519215725
x_65 = 0.26319289262190193
x_66 = 0.8112957298949761
x_67 = 0.5193420674823445
x_68 = 0.9423406150843128
x_69 = 0.29080230335182544
x_70 = 0.846560262782982
x_71 = 0.4575174292075856
x_72 = 0.9469184380631406
x_73 = 0.26328618984503466
x_74 = 0.8113555793084702
```

```
x_75 = 0.5192549932904637
x_76 = 0.9423655338310183
x_77 = 0.2907152242148706
x_78 = 0.8464585357330376
x_79 = 0.45770451199371165
x_80 = 0.9469399930776525
x_81 = 0.2632758816646311
x_82 = 0.8113465102220021
x_83 = 0.5192691464874282
x_84 = 0.942361651286677
x_85 = 0.2907289386794704
x_86 = 0.8464745706444899
x_87 = 0.45767502391388976
x_88 = 0.946936609494079
x_89 = 0.26327746974426125
x_90 = 0.8113478917438828
x_91 = 0.5192669949694847
x_92 = 0.9423622422915766
x_93 = 0.29072685177937274
x_94 = 0.8464721307706078
x_95 = 0.457679510895195
x_96 = 0.9469371246908698
x_97 = 0.26327722719648283
x_98 = 0.8113476803500026
x_99 = 0.5192673242983633
```

Actividad 8. Dados los elementos obtenidos en las actividades 4 y 7, obtener una lista que resulte de multiplicar los elementos de las dos lista dos a dos.

[8]:
```python
x = []

for i in range(len(x4)):
    x.append(x4[1] * x7[i])
    print(x[i])
```

```
0.49375
0.246875
0.6719628906250001
0.6978064343452455
0.7744408249631637
0.6694567784308072
0.8323587777811927
0.5547636689362824
0.9282888785540994
0.30358599380890783
0.8549523593865876
0.4158622526729143
0.9269057737955831
0.27246097755087745
```

0.8161871365794311
0.4875853107510914
0.935299961464251
0.2683644097805284
0.812796185030324
0.493086885397921
0.9345649145946546
0.2716855420622025
0.8170705457110401
0.4856800471788963
0.9354394515121555
0.2675184845633979
0.8116849546478143
0.4949989024801264
0.9342837468897389
0.27290696667389397
0.818623188116408
0.48296616409771653
0.9356741111787998
0.26621796840091255
0.8099732390838756
0.49793026879013436
0.9338080733294957
0.2748964309901304
0.8211285518273007
0.4785611466579108
0.9359572216592827
0.2643657935016116
0.8075051503421349
0.502131752776756
0.9330327137467019
0.2779922179041886
0.8249711043319646
0.4717429821113887
0.9361570370739422
0.26213097367021393
0.8044643266946112
0.5072697524776969
0.9319349180524784
0.2821665583934528
0.830048258726906
0.4626192706301162
0.9359718852484238
0.260341417779519
0.8019214600495809
0.5115435955776086
0.9308977163816569
0.28595870858130984

```
0.8345599542270731
0.4544025899445486
0.9353625200225529
0.25990298146412816
0.8011545332712889
0.5128502916388151
0.930561357395759
0.28716727455992763
0.8359782594981948
0.4517984613424908
0.9350819575873514
0.2599951124719717
0.8012136345671144
0.5127643058743329
0.9305859646581306
0.28708128391218474
0.8358778040363747
0.4519832055937903
0.9351032431641819
0.25998493314382326
0.8012046788442271
0.5127782821563354
0.9305821306455936
0.28709482694597704
0.8358936385114338
0.4519540861149662
0.935099901875403
0.259986501372458
0.8012060430970843
0.5127761575323662
0.9305827142629319
0.2870927661321306
0.8358912291359752
0.4519585170090051
0.9351004106322339
0.2599862618565268
0.8012058343456276
0.5127764827446338
```

Actividad 9. Definir las funciones siguientes.

$$f_1(x) = 3x^2 + x - 1 \qquad f_2(x) = \frac{2x+1}{x^2+1}$$

$$f_3(x) = \begin{cases} 2x & \text{si } x \leq 0 \\ x^2 & \text{si } x > 0 \end{cases} \qquad f_4(x) = \begin{cases} \dfrac{2x}{x+1} & \text{si } 0 < x \leq -2 \\ x^2 + 3 & \text{si } x > -2 \end{cases}$$

$$f_5(x) = \begin{cases} 2x & \text{si } x \leq 0 \\ x^2 & \text{si } 0 < x < 2 \\ x^3 + 1 & \text{si } x \geq 2 \end{cases} \qquad f_6(x) = \begin{cases} \dfrac{2x+1}{x^2} & \text{si } x \leq -1 \\ x^2 & \text{si } 0 < x < 2 \\ 0 & \text{si } x \geq 3 \end{cases}$$

[9]:
```
def f1(x):
    return 3 * x**2 + x - 1

# Definición de f2(x) = (2x + 1) / (x^2 + 1)
def f2(x):
    return (2 * x + 1) / (x**2 + 1)



def f3(x):
    if x <= 0:
        return 2 * x
    else:
        return x**2



def f4(x):
    if 0 < x <= -2:
        return (2 * x) / (x + 1)
    else:
        return x**2 + 3

def f5(x):
    if x <= 0:
        return 2 * x
    elif 0 < x < 2:
        return x**2
    else:
        return x**3 + 1



def f6(x):
    if x <= -1:
        return (2 * x + 1) / (x**2)
    elif 0 < x < 2:
        return x**2
    else:
```

```
        return 0
```

Actividad 12. Dados los conjuntos:

$$A = \{1, 2, 3, 4, 5\},$$
$$B = \{2, 4, 6, 8, 10, 12\}$$

y

$$C = \{1, 9, 4, 3, 2, 5, 11\},$$

obtener:

$a)$ $A \cap B \cup C$ $\quad$ $b)$ $B \backslash C \cup A$
$c)$ $(B \backslash C) \cup A$ $\quad$ $d)$ $(A \cup C) \triangle B$
$e)$ $A \cap (C \triangle B)$ $\quad$ $f)$ $(A \triangle B) \cup (B \backslash C)$

[10]:
```
A = {1,2,3,4,5}
B = {2,4,6,8,10,12}
C = {1,9,4,3,2,5,11}

print(f"a) {A & B | C}")
print(f"b) {B - C | A}")
print(f"c) {(B - C) | A}")
print(f"d) {(A | C) ^ B}")
print(f"e) {A & (C ^ B)}")
print(f"f) {(A ^ B) | (B - C)}")
```

```
a) {1, 2, 3, 4, 5, 9, 11}
b) {1, 2, 3, 4, 5, 6, 8, 10, 12}
c) {1, 2, 3, 4, 5, 6, 8, 10, 12}
d) {1, 3, 5, 6, 8, 9, 10, 11, 12}
e) {1, 3, 5}
f) {1, 3, 5, 6, 8, 10, 12}
```

Actividad 13. Crear un módulo llamado *fun1var.py* con las funciones definidas en la actividad 9.

[11]:
```
from fun1var import *
```

Actividad 17. Demostrar que las proposiciones $\neg(p \wedge q)$ y $\neg p \vee \neg q$ son lógicamente equivalentes.

[12]:
```
from sympy import symbols, And, Not, Or

p, q=symbols('p q')

Not(And(p, q)).equals(Or(Not(p), Not(q)))
```

[12]: True

Actividad 18. Demostrar que el argumento $\{p \to q, \neg p\}$ implica $\neg q$.

[13]:
```
from sympy import symbols, And, Not, Or, Implies

p, q=symbols('p q')
```

```
print(Implies(And(Implies(p, q), Not(p)), Not(q)).subs({p:True, q:True}))
print(Implies(And(Implies(p, q), Not(p)), Not(q)).subs({p:True, q:False}))
print(Implies(And(Implies(p, q), Not(p)), Not(q)).subs({p:False, q:True}))
print(Implies(And(Implies(p, q), Not(p)), Not(q)).subs({p:False, q:False}))
```

```
True
True
False
True
```

Actividad 19. Determinar la validez del argumento $\{p \to q, \neg p\}$ implica $\neg p$.

[14]:
```
from sympy import symbols, And, Not, Or, Implies

import sympy as sy

p, q=symbols('p q')

print(Implies(And(Implies(p, q), Not(p)), Not(p)).subs({p:True, q:True}))
print(Implies(And(Implies(p, q), Not(p)), Not(p)).subs({p:True, q:False}))
print(Implies(And(Implies(p, q), Not(p)), Not(p)).subs({p:False, q:True}))
print(Implies(And(Implies(p, q), Not(p)), Not(p)).subs({p:False, q:False}))
print()
print(sy.simplify_logic(Implies(And(Implies(p, q), Not(p)), Not(p))))
```

```
True
True
True
True

True
```

Actividad 20. Demostrar que el argumento $\{p \to \neg q, r \vee q, r\}$ implica $\neg p$

[15]:
```
from sympy import symbols, And, Not, Or, Implies, simplify_logic

p, q, r=symbols('p q r')

print(Implies(And(And(Implies(p, Not(q)), Or(r,q)), r), Not(p)).subs({p:True, q:
 ↪True, r:True}))
print(Implies(And(And(Implies(p, Not(q)), Or(r,q)), r), Not(p)).subs({p:True, q:
 ↪False, r:True}))
print(Implies(And(And(Implies(p, Not(q)), Or(r,q)), r), Not(p)).subs({p:True, q:
 ↪True, r:False}))
print(Implies(And(And(Implies(p, Not(q)), Or(r,q)), r), Not(p)).subs({p:False, q:
 ↪True, r:True}))
print(Implies(And(And(Implies(p, Not(q)), Or(r,q)), r), Not(p)).subs({p:False, q:
 ↪False, r:True}))
```

```
print(Implies(And(And(Implies(p, Not(q)), Or(r,q)), r), Not(p)).subs({p:False, q:
 ↪True, r:False}))
print(Implies(And(And(Implies(p, Not(q)), Or(r,q)), r), Not(p)).subs({p:True, q:
 ↪False, r:False}))
print(Implies(And(And(Implies(p, Not(q)), Or(r,q)), r), Not(p)).subs({p:False, q:
 ↪False, r:False}))
```

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

Actividad 21. Demostrar que $(p \vee q \wedge \neg r) \wedge (p \wedge q)$ es equivalente a $p \wedge q$.

[16]:
```
from sympy import simplify_logic, And, Or, Not, Implies, Equivalent, symbols

p, q, r = symbols('p q r')

And(And(Or(p, q), Not(r)), And(p, q)).equals(And(p, q))
```

[16]: `False`

Actividad 22. Demostrar que $(p \vee q \wedge \neg r) \wedge (p \vee q)$ es equivalente a $p \vee (q \wedge \neg r)$.

[17]:
```
from sympy import simplify_logic, And, Or, Not, Implies, Equivalent, symbols

p, q, r = symbols('p q r')

And(And(Or(p, q), Not(r)), Or(p, q)).equals(Or(p, And(q, Not(r))))
```

[17]: `False`

Actividad 23. Demostrar que $(p \vee q \wedge \neg r) \vee (p \vee q)$ es equivalente a $p \vee q$.

[18]:
```
from sympy import simplify_logic, And, Or, Not, Implies, Equivalent, symbols

p, q, r = symbols('p q r')

Or(And(Or(p, q), Not(r)), Or(p, q)).equals(Or(p, q))
```

[18]: `False`

Actividad 24. Simplficar los siguientes conjuntos.

a) $(A \cap B \cup C^c) \cap (C \cup A)$	*b*) $(A^c \cap B^c \cup C) \triangle (C \cup A)$
c) $(A \cap B \cup C^c) \cap ((C \cup A) \backslash B)$	*d*) $(A \cap B \cup C^c) \cap (C \cup A^c)$
e) $(A \cap B \cup C^c) \cup (C \cup A)^c$	*f*) $(A \cap B)^c \cap (C \cup A)^c \backslash (C \cup B)$

Nota: Se debe usar que $A \backslash B = A \cap B^c$ y $A \triangle B = (A \backslash B) \cup (B \backslash A)$

[19]:
```
from sympy import simplify_logic, symbols
A, B, C = symbols("A B C")

print(f"a) {simplify_logic((A & B | ~C) & (C | A))}")
print(f"b) {simplify_logic(((~A & ~B | C) & ~(C | ~A)) | ((C | A) & ~(~A & ~B |␣
↪C)))}")
print(f"c) {simplify_logic((A & B | ~C) & ((C | A) & ~B))}")
print(f"d) {simplify_logic((A & B | ~C) & (C | ~A))}")
print(f"e) {simplify_logic((A & B | ~C) | ~(C | A))}")
print(f"f) {simplify_logic(~(A & B) & ~(C | A) & ~(C | B))}")
```

```
a) A & (B | ~C)
b) A & ~C
c) A & ~B & ~C
d) (A | ~C) & (B | ~C) & (C | ~A)
e) ~C | (A & B)
f) ~A & ~B & ~C
```