Prácticas con Python

Francisco Javier Mercader Martínez

Actividad 1. Realizar las siguientes operaciones con Python:

(a) $(2^4+3)^2$

(b) $\dfrac{2+4^4}{1+\frac{2}{4\cdot 3^3}}$

(c) $\left(\dfrac{4+4^3}{2}+5^2\right)^6$

(d) $1+2\dfrac{7}{2^4+5}$

(e) $(2+3^2+5^3)^{\frac{1}{3}}$

(f) $\left(1+2^3\dfrac{5}{2^4+1}\right)^{\frac{1}{2}}$

```
(2**4+3)**2
(2+4**4)/(1+2/(4*3**3))
((4+4**3)/2+5**2)**6
(1+2*(7/(2**4+5)))
(2+3**2+5**3)**(1/3)
(1+2**3*5/(2**4+1))**(1/2)
```

$(2^4+3)^2 = 361$

$\dfrac{2+4^4}{1+\frac{2}{4\cdot 3^3}} = 253.3091$

$\left(\dfrac{4+4^3}{2}+5^2\right)^6 = 42180533641$

$1+2\dfrac{7}{2^4+5} = 1.6667$

$(2+3^2+5^3)^{\frac{1}{3}} = 5.1426$

$\left(1+2^3\dfrac{5}{2^4+1}\right)^{\frac{1}{2}} = 1.8311$

Activadad 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

```
print("45 entre 3")
print(f"Cociente: {45 // 3}")
print(f"Resto: {45 % 3}")
print()
print("111 entre 67")
print(f"Cociente: {111 // 67}")
print(f"Resto: {111 % 67}")
print()
print("99 entre 54")
print(f"Cociente: {99 // 54}")
print(f"Resto: {99 % 54}")
print()
print("103964 entre 78")
print(f"Cociente: {103964 // 78}")
print(f"Resto: {103964 % 78}")
```

```
45 entre 3
Cociente: 15
Resto: 0
```

```
111 entre 67
Cociente: 1
Resto: 44

99 entre 54
Cociente: 1
Resto: 45

103964 entre 78
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:

```
A = list((i+1)*2 for i in range(10))
B = list((i+1)*3 for i in range(5))
```

1. Hacer la unión de A y B. Llamar C a esta nueva lista

   ```
   C = list(set(A) | set(B))
   print(f"C = {C}")
   ```

   ```
   C = [2, 3, 4, 6, 8, 9, 10, 12, 14, 15, 16, 18, 20]
   ```

2. Eliminar los elementos repetidos de C eliminado el elemento repetido que aparece en primer lugar.

   ```
   C = list(dict.fromkeys(C))
   print(f"C sin repetidos (primer lugar): {C}")
   ```

   ```
   C sin repetidos (primer lugar): [2, 3, 4, 6, 8, 9, 10, 12, 14, 15, 16, 18, 20]
   ```

3. Añadir a la lista resultante los número 5 y 7 al final de la lista.

   ```
   C.extend([5, 7])
   print(f"C con 5 y 7 añadidos: {C}")
   ```

   ```
   C con 5 y 7 añadidos: [2, 3, 4, 6, 8, 9, 10, 12, 14, 15, 16, 18, 20, 5, 7]
   ```

4. Añadir a la lista resultante los números 3, 4 y 5 al principio de la lista.

   ```
   C = [3,4,5] + C
   print(f"C on 3, 4 y 5 al principio: {C}")
   ```

   ```
   C on 3, 4 y 5 al principio: [3, 4, 5, 2, 3, 4, 6, 8, 9, 10, 12, 14, 15, 16, 18, 20, 5, 7]
   ```

5. Eliminar los elementos repetidos de C eliminado el elemento repetido que aparece en último lugar.

   ```
   C = list(dict.fromkeys(C[::-1]))[::-1]
   print(f"C sin elementos repetidos (último lugar): {C}")
   ```

   ```
   C sin elementos repetidos (último lugar): [2, 3, 4, 6, 8, 9, 10, 12, 14, 15, 16, 18, 20, 5, 7]
   ```

6. Crear una nueva lista D con los elementos pares de C, sin escribir el número en cuestión, sino seleccionándolo de la lista C.

   ```
   D = [i for i in C if i % 2 == 0]
   print(f"D = {D}")
   ```

   ```
   D = [2, 4, 6, 8, 10, 12, 14, 16, 18, 20]
   ```

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

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

```
# Definir la función f(x)
def f(x):
    return 3.5 * x * (1-x)

# Inicilizar la lista con el valor inicial x_0
x_0 = 0.5
```

```
val_recursion = [x_0]

# Calcular los 100 primeros elementos de la recursión
for _ in range(100):
    x_next = f(val_recursion[-1])
    val_recursion.append(x_next)

# Imprimir los primeros 10 valores de la recursión para ver qué todo es correcto
for i, val in enumerate(val_recursion[:10]):
    print(f"x_{i} = {val}")
```

```
x_0 = 0.5
x_1 = 0.875
x_2 = 0.3828125
x_3 = 0.826934814453125
x_4 = 0.5008976948447526
x_5 = 0.87499717950388
x_6 = 0.3828199037744718
x_7 = 0.826940887670016
x_8 = 0.500883795893397
x_9 = 0.8749972661668659
```

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

```
# Definir la función f(x)
def f(x):
    return 3.5 * x * (1-x)

# Inicilizar la lista con el valor inicial x_0
x_0 = 0.5
val_recursion_pares = [x_0]

# Calcular los 200 primeros elementos de la recursión para luego hacer la separación
for _ in range(200):
    x_next = f(val_recursion_pares[-1])
    val_recursion_pares.append(x_next)

val_recursion_pares = val_recursion_pares[::2][:100]

# Imprimir los primeros 10 valores de la recursión para ver qué todo es correcto
for i, val in enumerate(val_recursion_pares[:10]):
    print(f"x_{2*i} = {val}")
```

```
x_0 = 0.5
x_2 = 0.3828125
x_4 = 0.5008976948447526
x_6 = 0.3828199037744718
x_8 = 0.500883795893397
x_10 = 0.38281967628581853
x_12 = 0.5008842229438679
x_14 = 0.3828196832226365
x_16 = 0.5008842099217973
x_18 = 0.3828196830110622
```

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}.

```
# Definir la función f(x)
def f(x):
    return 3.5 * x * (1-x)

# Inicilizar la lista con el valor inicial x_0
x_0 = 0.5
val_recursion_4 = [x_0]
```

```python
# Calcular los 400 primeros elementos de la recursión para luego hacer la separación
for _ in range(400):
    x_next = f(val_recursion_4[-1])
    val_recursion_4.append(x_next)

val_recursion_4 = val_recursion_4[::4][:100]

# Imprimir los primeros 10 valores de la recursión para ver qué todo es correcto
for i, val in enumerate(val_recursion_4[:10]):
    print(f"x_{4*i} = {val}")
```

```
x_0 = 0.5
x_4 = 0.5008976948447526
x_8 = 0.500883795893397
x_12 = 0.5008842229438679
x_16 = 0.5008842099217973
x_20 = 0.5008842103189732
x_24 = 0.5008842103068593
x_28 = 0.5008842103072292
x_32 = 0.5008842103072179
x_36 = 0.5008842103072179
```

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

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

```python
# Definir la función f(x)
def f(x):
    return 3.5 * x * (1-x)

# Inicilizar la lista con el valor inicial x_0
x_0 = 0.5
x_1 = 0.25
val_recursion_2 = [x_0, x_1]

# Calcular los 100 primeros elementos de la recursión para luego hacer la separación
for _ in range(100):
    x_next = 0.25 * val_recursion_2[-2] + 0.75*f(val_recursion_2[-1])
    val_recursion_2.append(x_next)

# Imprimir los primeros 10 valores de la recursión para ver qué todo es correcto
for i, val in enumerate(val_recursion_2[:10]):
    print(f"x_{i} = {val}")
```

```
x_0 = 0.5
x_1 = 0.25
x_2 = 0.6171875
x_3 = 0.6827011108398438
x_4 = 0.7229251732569537
x_5 = 0.6964742414218985
x_6 = 0.7356507085156734
x_7 = 0.6845990122426351
x_8 = 0.7507110894114075
x_9 = 0.66240262088179
```

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.

```python
val_recursion_3 = []

for i, val_1 in enumerate(val_recursion):
    for j, val_2 in enumerate(val_recursion_2):
        val_recursion_3.append(val_1 * val_2)
```

```
for i, val in enumerate(val_recursion_3[:10]):
    print(f"x_{i} = {val}")
```

```
x_0 = 0.25
x_1 = 0.125
x_2 = 0.30859375
x_3 = 0.3413505554199219
x_4 = 0.36146258662847686
x_5 = 0.34823712071094926
x_6 = 0.3678253542578367
x_7 = 0.34229950612131754
x_8 = 0.37535554470570376
x_9 = 0.331201310440895
```

Actividad 9. Definir las funciones siguientes

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

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

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

```
def f_1(x):
    return 3*x**2+x-1

def f_2(x):
    return (2*x+1)/(x**2+1)

def f_3(x):
    if x <= 0:
        return 2*x
    else:
        return x**2

def f_4(x):
    if x > 0 and x <= 2:
        return 2*x/(x+1)
    elif x > 2:
        return x**2+3

def f_5(x):
    if x <= 0:
        return 2*x
    elif x > 0 and x < 2:
        return x**2
    else:
        return x**3 + 1

def f_6(x):
    if x <= -1:
        return (2*x+1)/x**2
    elif x > 0 and x < 2:
        return x**2
    elif x >= 3:
        return 0
```

Actividad 10. Definir las funciones siguientes

$$f_1(x,y) = xy^2 \qquad f_2(x,y) = \frac{x+y^2}{x-y}$$
$$f_3(x,y,z) = xy^2 + zy^3 \qquad f_4(x,y,z,t) = x^2 + y^2 - z^{t-x}$$
$$f_5(x,y) = \begin{cases} 2xy & \text{si } xy \leq 0 \\ xy^2 & \text{si } xy > 0 \end{cases} \qquad f_6(x,y) = \begin{cases} 2x^y & \text{si } x+y^2 \leq 1 \\ x^{y^2} & \text{si } x+y^2 > 1 \end{cases}$$

```
def f1(x, y):
    return x * y**2

def f2(x, y):
    return (x + y**2) / (x - y)

def f3(x, y, z):
    return x * y**2 + z * y**3

def f4(x, y, z, t):
    return x**2 + y**2 - z**(t - x)

def f5(x, y):
    if x * y <= 0:
        return 2 * x * y
    else:
        return x * y**2

def f6(x, y):
    if x + y**2 <= 1:
        return 2 * x**y
    else:
        return x**(y**2)
```

Actividad 11. Dadas las lista A de los 10 primeros números naturales impares y B de los 5 primeros múltiplos de 4, hacer las siguientes operaciones:

```
A = [i for i in range(1, 20, 2)]
B = [(i+1)*4 for i in range(5)]
```

1. Insertar en A el número 10 en la posición 2 y llamar A a la lista resultante.

   ```
   A.insert(2, 10)
   print(A)
   ```

   ```
   [1, 3, 10, 5, 7, 9, 11, 13, 15, 17, 19]
   ```

2. Eliminar de B el primer y último elemento y llamar B a la lista resultante.

   ```
   B = B[1:-1]
   print(B)
   ```

   ```
   [8, 12, 16]
   ```

3. Añadir a A los dos primeros elementos de B y llamar A a la lista resultante.

   ```
   A.extend(B[:2])
   print(A)
   ```

   ```
   [1, 3, 10, 5, 7, 9, 11, 13, 15, 17, 19, 8, 12]
   ```

4. Definir C como la unión de B y A, por este orden.

   ```
   C = B + A
   print(C)
   ```

   ```
   [8, 12, 16, 1, 3, 10, 5, 7, 9, 11, 13, 15, 17, 19, 8, 12]
   ```

5. Añadir a la lista resultante los números 3,4 y 5 al final de la lista.

   ```
   C.extend([3,4,5])
   print(C)
   ```

```
[8, 12, 16, 1, 3, 10, 5, 7, 9, 11, 13, 15, 17, 19, 8, 12, 3, 4, 5]
```

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

```python
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.

Actividad 14. Crear un módulo llamado **`fun2var.py`** con las funciones definidas en la actividad 10.

Actividad 15. Construir con Python las tablas de verdad de $p \vee q$, $p \wedge q$, y $p \leftrightarrow q$.

```python
from sympy import symbols, Or, And, Equivalent

# Definición de las variables lógicas
p, q = symbols('p q')

# Lista de combinaciones de valores para p y q
combinaciones = [(True, True), (True, False), (False, True), (False, False)]

# Evaluación y conversión a 1 (True) o 0 (False)
for p_val, q_val in combinaciones:
    or_res    = Or(p, q).subs({p: p_val, q: q_val})
    and_res   = And(p, q).subs({p: p_val, q: q_val})
    equiv_res = Equivalent(p, q).subs({p: p_val, q: q_val})
```

p	q	$p \vee q$	$p \wedge q$	$p \leftrightarrow q$
1	1	1	1	1
1	0	1	0	0
0	1	1	0	0
0	0	0	0	1

Actividad 16. Comprobar que $p \vee \neg(p \wedge q)$ es una tautología.

```
from sympy import symbols, Or, And, Not

# Definición de las variables lógicas
p, q = symbols('p q')

# Lista de combinaciones de valores para p y q
combinaciones = [(True, True), (True, False), (False, True), (False, False)]

# Evaluación y conversión a 1 (True) o 0 (False)
for p_val, q_val in combinaciones:
    p_and_q = And(p, q).subs({p: p_val, q: q_val})
    not_p_and_q = Not(And(p, q)).subs({p: p_val, q: q_val})
    p_or_not_p_and_q = Or(p, Not(And(p, q))).subs({p: p_val, q: q_val})
```

p	q	$p \wedge q$	$\neg(p \wedge q)$	$p \vee \neg(p \wedge q)$
1	1	0	0	1
1	0	0	0	1
0	1	0	0	1
0	0	0	0	1

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

```
from sympy import symbols, Or, And, Not

p, q = symbols('p q')

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

```
True
```

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

```
from sympy import symbols, Or, And, Not, Implies

p, q = symbols('p q')

combinaciones = [(True, True), (True, False), (False, True), (False, False)]

# Evaluación y conversión a 1 (True) o 0 (False)
for p_val, q_val in combinaciones:
    p_to_q = Implies(p, q).subs({p: p_val, q: q_val})
    p_to_q_and_not_p = And(Implies(p, q), Not(p)).subs({p: p_val, q: q_val})
    p_to_q_and_not_p_implies_not_p = Implies(And(Implies(p, q), Not(p)),
                                             Not(q)).subs({p: p_val, q: q_val})
```

p	q	$p \to q$	$\neg p$	$\neg q$	$(p \to q) \wedge \neg p$	$((p \to q) \wedge \neg p) \to \neg q$
1	1	1	0	0	0	1
1	0	0	0	1	0	1
0	1	1	1	0	1	0
0	0	1	1	1	1	1

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

```
from sympy import symbols, Or, And, Not, Implies

p, q = symbols('p q')

combinaciones = [(True, True), (True, False), (False, True), (False, False)]

# Evaluación y conversión a 1 (True) o 0 (False)
for p_val, q_val in combinaciones:
    p_to_q = Implies(p, q).subs({p: p_val, q: q_val})
```

```
    p_to_q_and_not_p = And(Implies(p, q), Not(p)).subs({p: p_val, q: q_val})
    p_to_q_and_not_p_implies_not_p = Implies(And(Implies(p, q), Not(p)),
                                             Not(q)).subs({p: p_val, q: q_val})
```