Programación funcional y orden superior. Ejercicios resueltos

Introducción

Diseña funciones para los siguientes cálculos:

- La derivada de una función (derivable) en un punto
- El método de bipartición para calcular el cero de una función en un intervalo supuesto que...
- Newton-Raphson, partiendo de un punto y supuesto que...
- Dado el término general de una sucesión a_n de reales (que en realidad es una función $a : N \to R$), define la función que da la lista de términos a_i para $i \in \{a_1, \ldots, a_n\}$, aplicando la función a cada término de la lista $[1, \ldots, n]$ mediante la función `map`. Expresa la función "sumatorio", que suma los términos de una sucesión entre dos límites dados, esta vez, usando lambda expresiones.
- También podríamos querer cuadrantes con los que cumplen una propiedad y hallar el sumatorio:

$$\sum_{i \in \{k_1, \ldots, k_2\}, p(i)} a_i$$

Vamos con las soluciones a este pequeño apartado:

- La derivada de una función (derivable) en un punto

In [1]:
```
from math import sin, pi

def derivada(f, x, eps=0.001):
    return (f(x + eps) - f(x)) / eps

print(derivada(sin, pi/3), derivada(sin, pi/3, eps=0.000001))
```

```
0.4995566904000077 0.4999995669718871
```

La función anterior tiene el siguiente tipo:

```
derivada: (R -> R, R) -> R
```

Es decir, recibe como parámetros una función f y un real x y da como resultado otro real, $f'x)$.

Pero podríamos desear convertir una función en su derivada; esto es:

```
derivada: (R -> R) -> (R -> R)
```

Vamos con ello:

In [2]:
```
from math import sin, pi

def derivada(f, eps=0.001):
    def der_f(x):
        return (f(x + eps) - f(x)) / eps
    return der_f

print(derivada(sin)(pi/3))
print(derivada(sin, eps=0.000001)(pi/3))

print("......................")

# A lo mejor lo ves más claro así:

der_sin = derivada(sin)
print(der_sin(pi/3))

der_sin = derivada(sin, eps=0.000001)
print(der_sin(pi/3))
```

```
0.4995566904000077
0.4999995669718871
......................
0.4995566904000077
0.4999995669718871
```

- El método de bipartición para calcular el cero de una función en un intervalo supuesto que...

In [3]:
```
# Bipartición y Newton Raphson, resueltos en el apartado de funciones
```

- Dado el término general de una sucesión a_n de reales (que en realidad es una función $a : N \to R$), define la función que da la lista de términos a_i para $i \in \{a_1, \ldots, a_n\}$, aplicando la función a cada término de la lista $[1, \ldots, n]$ mediante la función `map`. Expresa la función "sumatorio", que suma los términos de una sucesión entre dos límites dados, esta vez, usando lambda expresiones.

In [4]:
```
def a(n):
    return 2*n+1

print(list(map(a, range(1, 11))))

sumatorio = lambda a, inf, sup: sum(map(a, range(inf, sup+1)))

print(sumatorio(a, 1, 11))
```

```
[3, 5, 7, 9, 11, 13, 15, 17, 19, 21]
143
```

- Variante: además de dar los límites del sumatorio, podríamos desear imponer un predicado:

$$\sum_{i=0, i \, es \, par}^{n} a_i$$

o no imponer dicha condición, en cuyo caso se acumulan todos los términos de la sucesión entre los límites indicados:

$$\sum_{i=0}^{n} a_i$$

In [5]:
```
def a(n):
    return 2*n+1

def es_par(n):
    return n%2==0

print(list(map(a, range(1, 11))))

def sumatorio_p(a, inf, sup, p=lambda x: True):
    return sum([a(i) for i in range(inf, sup + 1) if p(i)])

print(sumatorio_p(a, 1, 11))
print(sumatorio_p(a, 1, 11, es_par))
```

```
[3, 5, 7, 9, 11, 13, 15, 17, 19, 21]
143
65
```

Función factorial, de nuevo

La función factorial toma un parámetro y calcula el producto de los números desde 1 hasta el dato. Escribe esto con lambda expresiones: $f = \lambda n : \ldots$.

In [6]:
```
from functools import reduce

prod = lambda a, b: a*b
factorial = lambda n : reduce(prod, range(1, n+1))

print(factorial(100))
```

```
93326215443944152681699238856266700490715968264381621468592963895217599993229915608941463976156518286253697920827223758251185210916864000000000000000000000000
```

Nombres de persona

Dada una lista de nombres de persona, tenemos una función que selecciona los que tienen una longitud menor o igual a una cantidad, dada. Funciona así, por ejemplo:

```
>>>l = ['Ana', 'Marta', 'Patricia', 'Alba', 'Silvia', 'Gloria', 'Lara']
>>>short_names(l, 5), short_names(l, 3)
(['Ana', 'Marta', 'Alba', 'Lara'], ['Ana'])
```

Te pido que definas la función short_names así:

```
>>>sort_names = lambda lista, n: list(filter ...)
```

In [7]:
```
l = ['Ana', 'Marta', 'Patricia', 'Alba', 'Silvia', 'Gloria', 'Lara']

short_names = lambda lista, n: list(filter(lambda name: len(name)<=n, lista))

short_names(l, 5), short_names(l, 3)
```

Out[7]:
```
(['Ana', 'Marta', 'Alba', 'Lara'], ['Ana'])
```

Múltiplos

Define la función `select_multiplos(n, k)`, que genera los números desde 1 hasta n que son múltiplos de k. Hazlo usando listas por comprensión.

In [8]:
```
def select_multiplos(n, k):
    return [i for i in range(1, n+1) if i % k == 0]

select_multiplos(100, 7)
```

Out[8]:
```
[7, 14, 21, 28, 35, 42, 49, 56, 63, 70, 77, 84, 91, 98]
```

Función para filter

Supongamos definida la función siguiente, que comprueba si un número es primo:

```
def es_primo(n):
    i = 2
    while i*i <= n and (n%i!=0):
        i += 1
    return i*i>n
```

Define una instrucción que actúe como la siguiente,

```
>>> print(list(filter(es_primo, range(2, 100))))
```

pero usando una lista definida por comprensión:

```
>>> print([... for ... if ...])
```

In [9]:
```
def es_primo(n):
    i = 2
    while i*i <= n and (n%i!=0):
        i += 1
    return i*i>n

print(list(filter(es_primo, range(2, 100))))

print(".......................")

print([p for p in range(2, 100) if es_primo(p)])
```

```
[2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97]
.......................
[2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97]
```

Puntos y distancias

Dado un punto especial P, deseamos definir una función `dist`, que aplicada a P dé como resultado otra función, (le llamaremos `dist` $_P$), que aplicada a un punto q calcule la distancia (a P) de q:

$$dist_P(q) = |P, q|$$

Tenemos ahora una colección de puntos del plano, esto es, una lista de pares de reales. La llamamos, sencillamente "puntos". También tenemos un punto especial 'P'. Deseamos diseñar una función que ordene la lista de puntos de más cercano a más lejano a "P", esto es, de menor a mayor distancia a "P", usando la función "sort" predefinida.

In [10]:
```
import random
from math import sqrt

puntos = [(random.random(), random.random()) for i in range(10)]

def dist(P):
    def distQ(q):
        return sqrt((P[0]-q[0])**2 + (P[1]-q[1])**2)
    return distQ

def ordenar_dist(lista, P):
    lista.sort(key=dist(P))

print(puntos)
print()

P = (0.5, 0.5)

ordenar_dist(puntos, P)

print([(q, sqrt((P[0]-q[0])**2 + (P[1]-q[1])**2) + (P[1]-q[1])**2)
       for q in puntos])
```

```
[(0.6611993198675902, 0.07979904038150953), (0.9428896316854939, 0.2274723981047142), (0.0647600041275
2914, 0.3935874451953556), (0.8504761730103245, 0.3000603974025403), (0.3320971448146315, 0.01065170
6724651655), (0.9738356255803958, 0.04908152278020835), (0.8084366873202871, 0.3089180328350634?), (0.
5248094228505216, 0.07707671616149603), (0.13567152528662818, 0.4742006307565786), (0.9280063560112662
4, 0.2981704370000687)]

[((0.8084366873202871, 0.3089180328350634?), 0.3993421794890678), ((0.13567152528662818, 0.4742006307
565786), 0.3659064114995702?), ((0.8504761730103245, 0.3000603974025403), 0.4434723039049723), ((0.524
8094228505216, 0.07707671616149603), 0.6025144450664479), ((0.06476000412752914, 0.3935874451953556),
0.4593831188020095), ((0.6611993198675902, 0.07979904038150953), 0.6266280171105128), ((0.928006356012
2624, 0.2981704370000687), 0.5139419066953794), ((0.3320971448146315, 0.010651706724651655), 0.756813?
8016549?2), ((0.9428896316854939, 0.22747239810471422), 0.5942929468529232), ((0.9738356255803958, 0.049
08152278020835), 0.8574281329134831)]
```