Cronograma

- Programación Funcional
- Funciones de Orden Superior
  - Map
  - Reduce
  - Filter
- Ejercicios

Programación funcional

- La programación funcional es un paradigma en el que se programa en base a funciones.
  - Existen lenguajes de programación exclusivamente para este tipo de paradigma. Por ejemplo **Haskell**
  - También podemos desarrollar el paradigma en lenguajes de programación que no son exclusivamente para esto.
  - Perl, Python, JavaScript, Java 8
  - **Python** incluye varias características tomadas de los lenguajes funcionales como las funciones de orden superior y las funciones lambda

Programación funcional. Ventajas

- Los programas escritos en estilo funcional son más fáciles de testear y depurar.
- Por su característica modular facilita la computación concurrente
- El estilo funcional se lleva muy bien con grandes volúmenes de datos. Es propicio para crear de manera eficiente algoritmos y programas Big Data

Funciones de orden superior

- Funciones de orden superior son aquellas que pueden tomar otras funciones como argumento (parámetro) y pueden devolver funciones como resultado de su ejecución.
- En Python todo son objetos y las funciones no son una excepción a esta regla.

Funciones de orden superior (2)

- Funciones como parámetro
- La importancia de los paréntesis

```
def prueba(f):
    return f()

def porEnviar():
    return 2+2

print(prueba(porEnviar))
```

```
def prueba(f):
    return f()

def porEnviar():
    return 2+2

print(prueba(porEnviar()))
```

Funciones de orden superior (3)

- Retornando funciones

```
def saludar(lang):
        def saludar_es():
                print "Hola"
        def saludar_en():
                print "Hi"
        def saludar_fr():
                print "Salut"

        lang_func = {
                "es": saludar_es,
                "en": saludar_en,
                "fr": saludar_fr
                }

        return lang_func[lang]

f = saludar("es")
f()
```

Funciones de orden superior (4)

- ¿Qué almacena la variable f?
  - Una función: *print (f) → <function saludar_es at 0x7f13212565f0>*
  - *type(f)* → <type 'function'>
- ¿Cual función almacena la variable?: *saludar_es()*
- Si queremos pedir otra función debemos llamar a la función de orden superior.
  - *f = saludar("en")*
- No es necesario guardar el resultado de la función de orden para invocar a la función resultado:
  - *saludar("en")()* → *Hi*

Funciones de orden superior

```
def conversor(sis):
    def sis_bin(numero):
        print('dec:', numero, 'bin:', bin(numero))

    def sis_oct(numero):
        print('dec:', numero, 'oct:', oct(numero))

    def sis_hex(numero):
        print('dec:', numero, 'hex:', hex(numero))

    sis_func = {'bin': sis_bin, 'oct': sis_oct, 'hex': sis_hex}
    return sis_func[sis]


 # crea una instancia del conversor hexadecimal
conversorhex = conversor('hex')
conversorhex(100)
print()
# otra forma de usar el conversor. Convierte 9 dec a binario
conversor('bin')(9)
```

Funciones de orden superior para listas

- **Problema:**

  lista_numeros = list (range(10))

  ¿Cómo hacemos para convertir nuestra lista de números a una lista de strings?

- **Solución I**
  - A cada elemento de nuestra lista debemos realizar la conversión uno a uno.

    for i in lista_numeros:

    lista_numeros[i] = str(i)

Funciones de orden superior para listas (2)

- **Python** nos provee 3 funciones muy importantes para sustituir los bucles típicos mediante iteraciones de orden superior (ventaja: eficiencia).
- Estas funciones son:
  - **map(function, sequence[, sequence, ...]) → iterator**
  - **reduce(function, sequence[, initial]) → value**
  - **filter(function or None, sequence) → iterator**

Función map

- **map**(function, sequence[, sequence, ...]) → *iterator*
  - Retorna un iterador donde cada elemento es el resultado de aplicar la función "*function*" en la secuencia pasada como parámetro (*sequence*)
  - Hay que castear el resultado a lista si se quiere tratar como tal (diferencia con PY 2.x)
  - Si se pasan como parámetros n secuencias, la función tendrá que aceptar n argumentos.
  - Si las listas tienen diferentes tamaños, el valor que le llega a la función "*function*" para las posiciones mayores (que no tienen su correspondiente en la otra lista de tamaño menor), será None .

Función map (2)

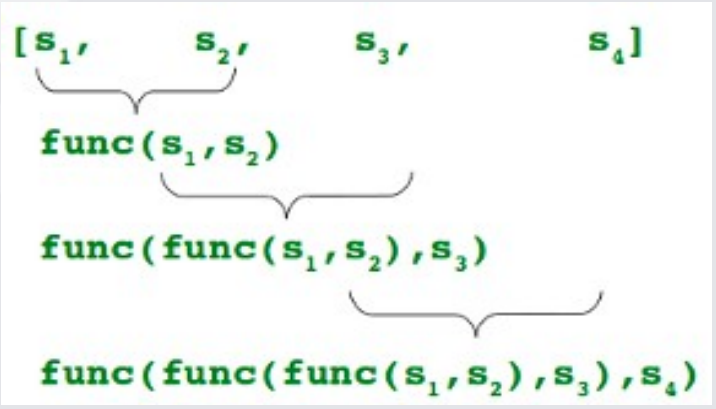
- **Solución II:**
  - *lista_num* → [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
  - *lista_str = list(map(str, lista_num))*
  - *lista_str* → ['0', '1', '2', '3', '4', '5', '6', '7', '8', '9']
- **Ejemplo: Sumar 2 listas elemento a elemento.**
  - *def sumar(x,y):*

    return x+y
  - *lista1, lista2 = range(10), range(10)*
  - *list(map(sumar,lista1, lista2))* → [0, 2, 4, 6, 8, 10, 12, 14, 16, 18]

Función reduce

- reduce(function, sequence[, initial]) → value
  - Se aplica la función "*function*" de a pares de elementos de una secuencia, hasta dejarla en un solo valor.
  - La función retorna un único valor que se construye llamando a la función "*function*" con los primeros dos ítems de la secuencia, luego se toma el resultado de estos dos primeros y el siguiente ítem, y así sucesivamente de izquierda a derecha.
- Se debe importar la

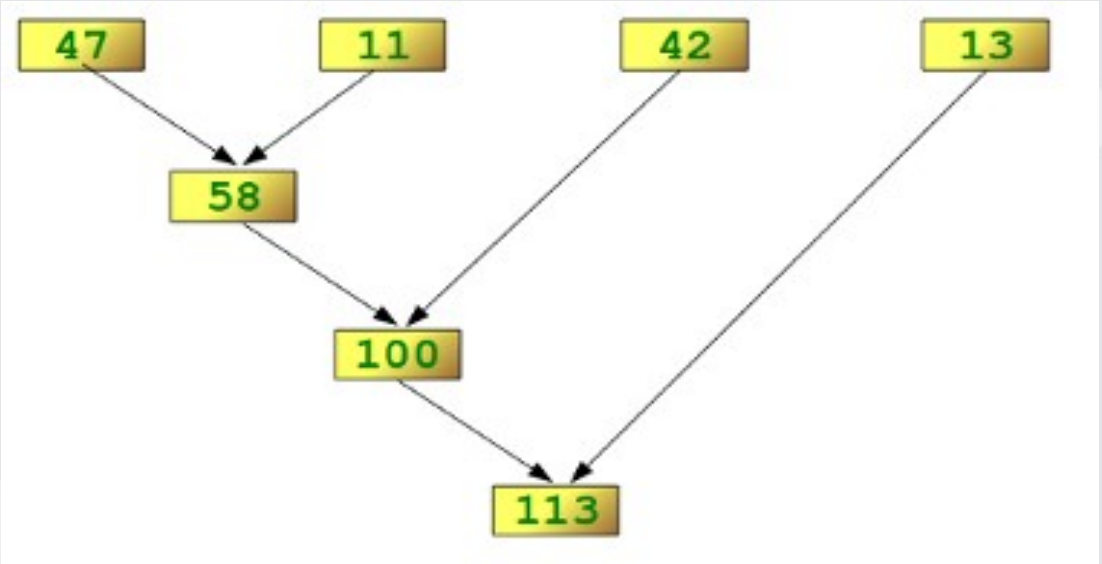
  librería **functools**



Función reduce (2)

import functools

functools.reduce(sumar, [47,11,42,13]) → 113



Función reduce (3)

- El parámetro opcional de reduce es un **inicializador**, esto significa que reduce evaluará la función de reducción comenzando por el valor especificado. Si no se especifica, el inicializador será el 1er. valor de la secuencia
- Ejemplo:

```
import functools
# Instead of:
product = functools.reduce(operator.mul, [1,2,3], 1)

# You can write:
product = 1
for i in [1,2,3]:
    product *= i
```

Función reduce (4)

- Ejemplos: Queremos calcular la suma de los números del 1 al 10

```
import functools
def sumar(x,y):
    return x+y
print(functools.reduce(sumar, (range(1, 11)))) → 55
```

- Queremos pasar de una lista de números impares a una palabra.

```
def concatenar(x,y):
    return x+y
functools.reduce(concatenar,  (map(str,range(1,15,2)))) →
135791113
```

Función filter

filter(function or None, sequence) → iterator

- Retorna los ítems de la secuencia para aquellos elementos que hagan verdadera a la función “function”.
- En el caso de que se pase como función None se devuelve la secuencia con los elementos que sean evaluados como verdadero.
- El tipo de dato que se devuelve es el mismo que se pasa como parámetro en sequence.

Función filter (2)

- Conservar solo los números que son pares.

```
def es_par(n):
    return (n % 2 == 0)
l = [1, 2, 3]
l2 = list(filter(es_par, l)) → [2]
```

- Obtener si el número es divisible entre 2 y 3

```
def f(x):
    return x % 2 != 0 and x % 3 != 0
print (filter(f, range(2, 25)))
print (list(filter(f, list(range(2, 25)))))
```

Ejercicios

- **Ejercicio 1:**
  - Utilizando la función reduce, escribir una función que tenga como entrada una lista de palabras y retorne la palabra mas larga.
  - Ídem pero que retorne la menor alfabéticamente
- **Ejercicio 2:**
  - Utilizando la función map, escribir una función que dada una lista de palabras retorne una lista de tuplas donde cada tupla tenga este formato: *(palabra, largo de la palabra)*

Ejercicios (2)

- **Ejercicio 3:**
  - Utilizando la función **filter**, escribir una función que tenga como entrada una lista de palabras y y retorne una lista con las palabras de largo mayor a 5.
- **Ejercicio 4:**
  - Usar filter para que, dado un string, se devuelva una lista con todas las vocales contenidas en él

Ejercicios (3)

Ejercicio 5

En una biblioteca tenemos la siguiente información acerca de órdenes de compra.

Id	Título y autor	Cantidad	Precio/unidad
34587	Learning Python, Mark Lutz	4	40.95
98762	Programming Python, Mark Lutz	5	56.80
77226	Head First Python, Paul Barry	3	32.95

Escribir un programa que retorne una lista con tuplas de 2 elementos. Cada tupla consiste en el id de la orden (primer elemento) y el precio (de acuerdo a la cantidad). El precio final debe tener un descuento de 10% en caso de que la orden supere los 100. La información relativa a las ordenes está representada como una lista de listas:

```
orders = [ ["34587", "Learning Python, Mark Lutz", 4, 40.95],
           ["98762", "Programming Python, Mark Lutz", 5, 56.80],
           ["77226", "Head First Python, Paul Barry", 3,32.95]]
```