Fundamentos da Programação
LEIC/LETI

Funções de ordem superior

Funções como parâmetros. Funções como valor.

Aula 26

Alberto Abad, Tagus Park, IST, 2021-22

Funções de ordem superior

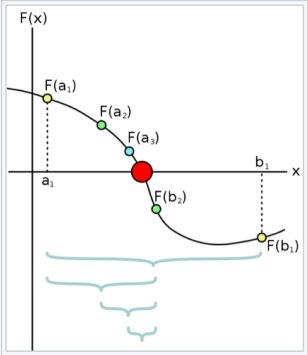
Funções de ordem superior

- Em aulas anteriores vimos que as funções permitem-nos abstrair algoritmos e procedimentos de cálculo (abstracção procedimental).
- Em Python, tal como nas linguagens puramente funcionais, as funções são entidades de primeira ordem/classe (*first class*):
  - Podemos nomear, utilizar como parâmetro e retornar como valor.
- Isto significa que podemos expressar certos padrões de computação geral a través de funções que manipulam outras funções, conehcios como **funções de ordem superior**:
  - Funções como parâmetros:
    - Funções como métodos gerais (hoje)
    - Funcionais sobre listas (próximo dia)
  - Funções como valor (hoje)

Funções como parâmetros - Funções como métodos gerais

Método da Bissecção

- O método da bissecção (https://en.wikipedia.org/wiki/Bisection_method) (baseado no Teorema Bolzano (https://en.wikipedia.org/wiki/Intermediate_value_theorem)) permite-nos obter a raíz de uma função contínua $f(x)$ situada no intervalo $[a, b]$, sempre que $f(a) <= 0 <= f(b)$ ou $f(b) <= 0 <= f(a)$:



A few steps of the bisection method applied over the starting range [a_1;b_1]. The bigger red dot is the root Wipe of the function.

Funções de ordem superior

Funções como parâmetros - Funções como métodos gerais

Método da Bissecção

In [136]:
```
def metodo_bisseccao(f, a, b):
    # Fazer, primeiro iterativa, logo recursiva
    def aproxima_raiz(f, a, b):
        m = (a + b)/2
        while (abs(f(m)) > 0.000001):
            if f(m) > 0:
                a = m
            else:
                b = m
            m = (a + b)/2
        return m

    if a > b:
        a, b = b, a

    x = f(a)
    y = f(b)
    if y < 0 < x:
        return aproxima_raiz(f, a, b)
    elif x < 0 < y:
        return aproxima_raiz(f, b, a)
    else:
        raise ValueError("metodo_bisseccao: sig(f(a)) == sig(f(b))!
?")


print(metodo_bisseccao(lambda x:x**3 - 2*x, -1, -10))
from math import sqrt
print(sqrt(2))

# from math import sin
# metodo_bisseccao(sin, 2, 4)
#sin(4)
```

```
-1.4142135381698608
1.4142135623730951
```

Funções de ordem superior

Funções como valor de funções - Potência geral

- As funções também podem produzir/retornar valores que são funções.

In [166]:
```
def make_power_of(n):
    def funcao_auxiliar(x):
        return x**n

    return funcao_auxiliar

make_power_of(3)(25)
```

Out[166]: 15625

- Reparem que neste exemplo o valor do expoente está ligado à função devolvida mesmo após o fim da chamada a `make_power_of` .
- Este tipo de técnica em que uma função mantém valores de *scopes* onde estava encapsulada não estando estes já presentes em memória, em Python e programação funcional é conhecida como **closure**.

Funções de ordem superior

Funções como valor de funções - Cálculo derivada

- As funções também podem produzir/retornar valores que são funções.
- Consideremos o cálculo da derivada de uma função de variável real `f` .
  - Por definição:

$$f'(a) = \lim_{x \to a} \frac{f(x) - f(a)}{x - a}$$

  - Substituindo $h = x - a$,

$$f'(a) = \lim_{h \to 0} \frac{f(a + h) - f(a)}{h}$$

  - Se dx for um número suficientemente pequeno, podemos considerar a seguinte aproximação:

$$f'(a) \approx \frac{f(a + dx) - f(a)}{dx}$$

Funções como valor de funções - Cálculo derivada

- Definamos primeiro a função de derivada num ponto:

$$f'(a) \approx \frac{f(a+dx) - f(a)}{dx}$$

In [152]:
```
def derivada_num_ponto(f, a):
    delta = 0.00001
    return (f(a+delta) - f(a))/delta

derivada_num_ponto(lambda x : x**2, 10)
```

Out[152]:
```
20.00000999942131
```

Funções de ordem superior

Funções como valor de funções - Cálculo derivada

- Podemos no entanto definir a função de **ordem superior** que retorna a derivada de f da seguinte forma (utilizando funções internas):

In [164]:
```
def derivada(f):
    def derivada_num_ponto(a):
        delta = 0.00000001
        return (f(a+delta) - f(a))/delta

    return derivada_num_ponto

derivada(lambda x: x*x*x +3*x - 1)(5)
```

Out[164]:
```
78.00000219049252
```

Funções como valor de funções - Cálculo derivada

- Podemos definir a mesma função utilizando funções *lambda* :

In [21]:
```
def derivada_lambda(f):
    delta = 0.000001
    return lambda x:(f(x+delta) - f(x))/delta

g = derivada_lambda(lambda x: x*x)
g(3)
```

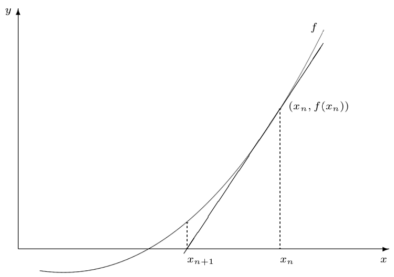
Out[21]: 6.000001000927568

Funções de ordem superior

Funções como valor de funções - Método de Newton

- Método para determinar raízes de funções diferenciáveis:
  - Partir de uma aproximação, x_n, para a raiz de uma função f,
  - Calcular, nova aproximação: $x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$
- A função matemática que calcula uma nova aproximação é chamada transformada de Newton:

$$t_{Newton}(x) = x - \frac{f(x)}{f'(x)}$$



In [115]:
```
def transformada_newton(f):
    def t_n(x):
        return x - f(x)/derivada(f)(x)
    return t_n

transformada_newton(lambda x:x*x)(-1)
```

Out[115]:
```
-0.4999974999850175
```

Funções de ordem superior

Funções como valor de funções - Método de Newton

In [109]:
```
def calcula_raizes(f, palpite):
    def bom_palpite(x):
        return abs(x) < 0.00001

    tf_N = transformada_newton(f)
    while not bom_palpite(f(palpite)):
        palpite = tf_N(palpite)

    return palpite


calcula_raizes(lambda x : x * x * x - 2 * x, 1)

# from math import sin
# calcula_raizes(sin, 2)
```

Out[109]:
```
1.4142142440881034
```

Tarefas próxima aula

- Estudar matéria de funções de ordem superior:
  - Completar exemplos
- Próxima aula teórica:
  - Projeto 2
  - Tópicos de Python: Exepções, GUI, etc.
  - Perspetiva sobre a próximas cadeiras do curso
- **Ficha 6** sobre **funcionais sobre listas** + **outro**

In [ ]: