Funções

Evitando repetições

- Em computação sempre queremos evitar repetições.
- Várias linguagens de programação incluem o conceito de subprograma (ou subrotina)
  - Atribui-se um nome à uma seqüência de comandos, e faz-se referência a este nome nos vários lugares do programa onde a seqüência em questão deveria ser repetida.
- Em Python, sub-programas têm o nome de ***funções***
- Formato geral:

```
def nome (arg, arg, ... arg):
    comando
    . . .
    comando
```

Definindo funções

- Em Python, sub-programas têm o nome de ***funções***
- Formato geral:

  `def` *nome* (*arg*, *arg*, ... *arg*):

  comando

  . . .

  comando

- Onde:
  - *nome* é o nome da função
  - *args* são especificações de argumentos da função
    - Uma função pode ter nenhum, um ou mais argumentos
  - *comandos* contêm as instruções a serem executadas quando a função é invocada

Resultado de funções

- Uma função tipicamente computa um ou mais valores
- Para indicar o valor a ser devolvido como o resultado da função, usa-se o comando `return`, que tem o formato

  `return` *expressão*

  - onde a *expressão* é opcional *e* designa o valor a ser retornado
- Ao encontrar o comando `return`, a função termina imediatamente e o controle do programa volta ao ponto onde a função foi chamada
  - observe que pode haver mais de um return dentro da função
- Se uma função chega a seu fim sem nenhum valor de retorno ter sido especificado, o valor de retorno é `None`

Exemplo

```
>>> def f():
        return

>>> print f()
None
>>> def f():
        return "Oi"

>>> print f()
Oi
>>> def f(nome):
        return "Oi, " + nome + "!"

>>> f("Joao")
Oi, Joao!
```

Argumentos de funções

- Argumentos (ou parâmetros) são como variáveis que recebem seus valores iniciais do chamador
- Essas variáveis, assim como outras definidas dentro da função são ditas *locais*, isto é, só existem no lugar onde foram definidas
  - Ao retornar ao ponto de chamada, as variáveis locais são descartadas
- Se uma função define *n* argumentos, a sua chamada deve incluir valores para todos eles
  - Exceção: argumentos com valores default

Exemplo

```
>>> def quadrado(x):
        return x*x

>>> quadrado(10)
100
>>> x
....
NameError: name 'x' is not defined
>>> quadrado()
....
TypeError: quadrado() takes exactly 1
  argument (0 given)
```

Variáveis Locais

- As variáveis de uma função NÃO tem NENHUMA relação com as variáveis da outra (o programa também é uma função). Ou seja, se você mencionar uma variável na função1 declarada na função2 irá dar erro. Outra conseqüência é que podem existir variáveis com mesmo nome em funções diferentes e nada terão em comum (além do nome).
- O valor de uma variável que pertença a uma determinada função morre quando essa função termina. Ou seja, toda vez que uma função é chamada é como se ela estivesse sendo executada pela primeira vez.
- Argumentos também são variáveis locais e valem as mesmas regras.
- Evitem variáveis globais!

Alterando parâmetros

- É possível alterar parâmetros?
  - Sim e não
  - Como o parâmetro é uma variável local, ele pode ser alterado sem problemas
  - Entretanto, se um parâmetro recebe um valor que vem de uma variável, esta não é alterada
- Ex.:

```
>>> def quadrado(x):
        x = x*x
        return x
>>>  x = 10
>>> quadrado(x)
100
>>> x
10
```

Retorno de mais de um valor

- Python permite o retorno de mais de um valor.
- Exemplo:

```
>>> def minmax(x,y):
        if x<y:
          return x,y
        else:
           return y,x

>>> a = 5
>>> b = 10
>>> minmax(a,b)
>>> (5, 10)
```

Ate aqui...!!

Observações

- A declaração da função deve ser feita antes da sua chamada na outra função para que o Python reconheça o seu nome.
- Os valores passados como parâmetros na função chamadora, serão recebidos pela função chamada **exatamente** na mesma ordem.
- Uma função pode chamar a si própria, isso é chamado de função recursiva. Cada vez que a função é chamada, uma nova instância de suas variáveis e parâmetros é criada.
- Não pode haver funções e variáveis com o mesmo nome!

Usando Função para “chamar” códigos.

```
#Série onde o próximo valor é a soma dos dois anteriores.

#ENTRADA
ant = 0 #número a ser impresso
atual = 0  #número anterior da série
prox = 1  #próximo número da série.
vlr_maximo =int(input('Qual o valor máximo = '))

#PROCESSAMENTO
while prox < vlr_maximo:
    print (prox)
    ant = atual
    atual = prox
    prox = ant + atual
```

Usando Função para “chamar” códigos.

```
def fibonacci():
    #Série onde o próximo valor é a soma dos dois anteriores.

    #ENTRADA
    ant = 0 #número a ser impresso
    atual = 0  #número anterior da série
    prox = 1  #próximo número da série.
    vlr_maximo =int(input('Qual o valor máximo = '))

    #PROCESSAMENTO
    while prox < vlr_maximo:
        print (prox)
        ant = atual
        atual = prox
        prox = ant + atual
```

Exercício...

- Utilize o FOR para implementar as funções abaixo:

1) Faça uma função para exibir os números pares de 10 a 50.

2)Faça uma função para exibir os números de 20 a 0.

3)Faça uma função para somar os números de 3 a 10.

4)Faça uma função para somar os números pares de 4 a 20.

5) Faça uma função para somar os números ímpares de 10 a 15.

6) Faça uma função para calcular a média dos números de 1 a 10.

Fim...