# MC-102 — Aula 14
# Funções I

Eduardo C. Xavier

Instituto de Computação – Unicamp

3 de Novembro de 2020

# Roteiro

# Funções

- Um ponto chave na resolução de um problema complexo é conseguir "quebrá-lo"em subproblemas menores.
- Ao criarmos um programa para resolver um problema, é crítico quebrar um código grande em partes menores, fáceis de serem entendidas e administradas.
- Isto é conhecido como modularizacão, e é empregado em qualquer projeto de engenharia envolvendo a construção de um sistema complexo.

# Funções

- Um conceito muito comum de modularização em linguagens de programação são funções:

## Função

Estrutura da linguagem que agrupa um conjunto de comandos que realizam uma tarefa específica. Uma função recebe argumentos como entrada, realiza uma computação e em geral devolve uma resposta para o invocador da função.

# Funções

- Vocês já usaram algumas funções como **input** e **print**.
- Algumas funções podem devolver algum valor ao final de sua execução:

```
n = float(input())
x = math.sqrt(n)
print(x)
```

- Este exemplo usa 4 funções: **input, float, sqrt, print**.
- Somente **print** não devolve nenhum valor.
- Vamos aprender como criar nossas próprias funções.

# Porque utilizar funções?

- Evitar que os blocos do programa fiquem grandes demais e, por conseqüência, mais difíceis de ler e entender.
- Separar o programa em partes que possam ser logicamente compreendidas de forma isolada.
- Permitir o reaproveitamento de código já construído (por você ou por outros programadores).
- Evitar que um trecho de código seja repetido várias vezes dentro de um mesmo programa, minimizando erros e facilitando alterações.

# Definindo uma função

Uma função é definida da seguinte forma:

```
def nome(parâmetro1,..., parâmetroN):
    comandos
    return valor_de_retorno
```

- Os **parâmetros** são variáveis, que são inicializadas com valores indicados durante a invocação da função (estes valores são os argumentos da função).
- O comando **return** devolve para o invocador da função o resultado da execução desta.

# Definindo uma função: Exemplo 1

A função abaixo recebe como parâmetro dois valores inteiros. A função faz a soma destes valores, e devolve o resultado.

```
def soma(a, b):
    c = a + b
    return c
```

- Quando o comando **return** é executado, a função para de executar e retorna o valor indicado para quem fez a invocação (ou chamada) da função.

# Definindo uma função: Exemplo 1

- Pode-se invocar a função passando como parâmetro dois valores inteiros, que serão associados com as variáveis **a** e **b** respectivamente.

```
def soma (a, b):
  c = a + b
  return c

r = soma(12, 10)
print("r = ", r)
r = soma(-9, 11)
print("r = ", r)
```

# Exemplo de função 1

```
def soma (a, b):
  c = a + b
  return c

r = soma(12,10)
print("r = ", r)
r = soma(-9, 11)
print("r = ", r)
```

- O programa faz a definição da função **soma** e em seguida executa os demais comandos.
- Quando se encontra a chamada para uma função, o fluxo de execução passa para ela e é executado os comandos até que um **return** seja encontrado ou o fim da função seja alcançado.
- Depois disso o fluxo de execução volta para o ponto onde a chamada da função ocorreu.

# Exemplo de função 1

- O que será impresso pelo programa abaixo?

```
def soma(a, b):
  c = a + b
  return c
  print("Bla bla bla!")

x1 = 3
x2 = 2
res = soma(x1, x2)
print("Soma é: ", res)
```

# Exemplo de função 1

- A expressão contida dentro do comando **return** é chamado de valor de retorno (é a resposta da função). Nada após ele será executado.

```
def soma(a, b):
  c = a + b
  return c
  print("Bla bla bla!")

x1 = 3
x2 = 2
res = soma(x1, x2)
print("Soma é: ", res)
```

- Não será impresso *Bla bla bla!*, somente será impresso o valor da soma.

# Definindo uma função: Exemplo 2

- A lista de parâmetros de uma função pode ser vazia:

```
def leNumeroInt():
  c = input("Digite um número inteiro: ")
  return int(c)


r = leNumeroInt()
print("Número digitado: ", r)
```

# Funções que retornam nada

- Em alguns casos faz sentido para uma função não retornar nada.
- **None** é um objeto em Python que representa o "nada".
- Há dois modos de criar funções que não retornam nada:
  - Use o comando **return None**.
  - Não use o comando **return** na função. Por padrão será devolvido **None** no fim da execução da função.

```
def imprime(num):
    print("Número: ", num)

x = imprime(5)
print(x) #será impresso None aqui
```

# Invocando uma função

- Na chamada da função, para cada um dos parâmetros desta, devemos fornecer um argumento que pode ser uma variável ou uma constante.
- Neste exemplo a função possui dois parâmetros, e na sua invocação são passados dois valores constantes inteiros:

```
def quadradoDaSoma(a, b):
    a = (a+b)*(a+b)
    return a

r = quadradoDaSoma(2, 2)
print(r)  #imprime 16
```

- Neste outro exemplo são passados dois valores associados às variáveis:

```
def quadradoDaSoma(a, b):
    a = (a+b)*(a+b)
    return a

a = 2
c = 3
r = quadradoDaSoma(a, c)
print(r)  #imprime 25
```

# Invocando uma função

```
def quadradoDaSoma(a, b):
    a = (a+b)*(a+b)
    return a

a = 2
c = 3
r = quadradoDaSoma(a, c)
print(r)  #imprime 25
```

- Um outro ponto importante deste exemplo é que as variáveis criadas dentro da função (incluindo as variáveis parâmetros) são **locais** à função.
- A variável **a** fora da função é uma variável diferente daquela declarada no parâmetro da função.
- Veremos mais sobre isso quando falarmos sobre variáveis locais e globais.

# Definindo funções depois do seu uso

- Até o momento, aprendemos que devemos definir as funções antes do seu uso. O que ocorreria se declarássemos depois?

```
x1 = leNumero()
x2 = leNumero()
res = soma(x1, x2)
print("Soma é: ", res)


def soma(a, b):
  c = a + b
  return c

def leNumero():
  c = int(input("Digite um número: "))
  return c
```

- Ocorre um erro ao executarmos o programa!

```
Traceback (most recent call last):
  File "t2.py", line 2, in <module>
    x1 = leNumero()
NameError: name 'leNumero' is not defined
```

# Definindo funções depois do seu uso

- É comum criarmos um função **main()** que executa os comandos iniciais do programa.
- O seu programa conterá então várias funções (incluindo a **main()**) e um único comando no final do arquivo que é a chamada da função **main()**.
- O programa será organizado da seguinte forma:

```
import bibliotecas

def main():
  Comandos Iniciais

def fun1(Parâmetros):
  Comandos

def fun2(Parâmetros):
  Comandos

...
...
main()
```

# Definindo funções depois do seu uso

Exemplo:

```
def main():
  x1 = leNumero()
  x2 = leNumero()
  res = soma(x1, x2)
  print("Soma é: ", res)


def soma(a, b):
  c = a + b
  return c

def leNumero():
  c = int(input("Digite um número: "))
  return c

main()
```

Agora a execução do programa ocorre sem problemas.

# Exemplo Utilizando Funções

- Em uma das aulas anteriores vimos como testar se um número candidato em **cand** é primo:

```
eprimo = True
for div in range(2,cand//2 + 1):
    if cand % div == 0:
        eprimo = False
        break

if eprimo:
    print(cand)
```

# Exemplo Utilizando Funções

- Depois usamos este código para imprimir os $n$ primeiros números primos:
- Veja no próximo slide.

# Exemplo Utilizando Funções

```
n = int(input('Quantidade de primos:'))

primos_impressos = 0
cand = 2
while primos_impressos < n:
    eprimo = True
    for div in range(2, cand//2 + 1):
        if cand % div == 0:
            eprimo = False
            break

    if eprimo:
        print(cand)
        primos_impressos = primos_impressos + 1
    cand = cand + 1
```

# Exemplo Utilizando Funções

- Podemos criar uma função que testa se um número é primo ou não (note que isto é exatamente um bloco que realiza uma computação bem definida).
- Depois fazemos chamadas para esta função.

# Exemplo Utilizando Funções

```
def ePrimo(cand):
    for div in range(2, cand//2 + 1):
        if cand % div == 0:
            return False
    #se terminou o laço necessariamente é primo
    return True
```

# Exemplo Utilizando Funções

- Suponha que precisamos de uma lista com os $n$ primeiros primos.
- Podemos criar uma função para devolver tal lista:

```
def Nprimos(n):
    r = []
    n_primos = 0
    cand = 2
    while n_primos < n:
        if ePrimo(cand):
            r.append(cand)
            n_primos += 1
        cand += 1
    return r
```

# Exemplo Utilizando Funções

```
def ePrimo(cand):
    for div in range(2, cand//2 + 1):
        if cand % div == 0:
            return False
    #se terminou o laço necessariamente é primo
    return True

def Nprimos(n):
    r = []
    n_primos = 0
    cand = 2
    while n_primos < n:
        if ePrimo(cand):
            r.append(cand)
            n_primos += 1
        cand += 1
    return r

def main():
    n = int(input('Quantidade de primos:'))
    print(Nprimos(n))
main()
```

# Exercício

- Escreva uma função que computa o fatorial de um número $n$ passado por parâmetro. Sua função deve ter o seguinte protótipo:

  **def fat(n):** OBS: Caso $n \leq 0$ seu programa deve retornar 1.

- Use a função anterior e crie um programa que imprima os valores de $n!$ para $n = 1, \ldots, 20$.

# Exercícios

- Escreva uma função que recebe uma matriz de inteiros como parâmetro e devolve uma lista contendo os elementos de menor e maior frequência de ocorrência na matriz. Assuma que os valores da matriz são inteiros entre 0 e 100.

# Exercícios

- Escreva uma função que recebe duas matrizes de floats como parâmetro e devolve uma terceira matriz que é o resultado da soma das duas matrizes passados por parâmetro.

# Exercícios

- Escreva uma função que recebe duas matrizes de floats como parâmetro e devolve uma terceira matriz que é o resultado da multiplicação das duas matrizes passados por parâmetro.