INSTITUTO FEDERAL
Rio Grande do Norte | Campus Parnamirim

Funções em Python

Adaptado do professor Cláudio Esperança

Prof. Fábio Procópio

Prof. João Nascimento

Técnico Integrado em Informática – Programação Estruturada e Orientada a Objetos

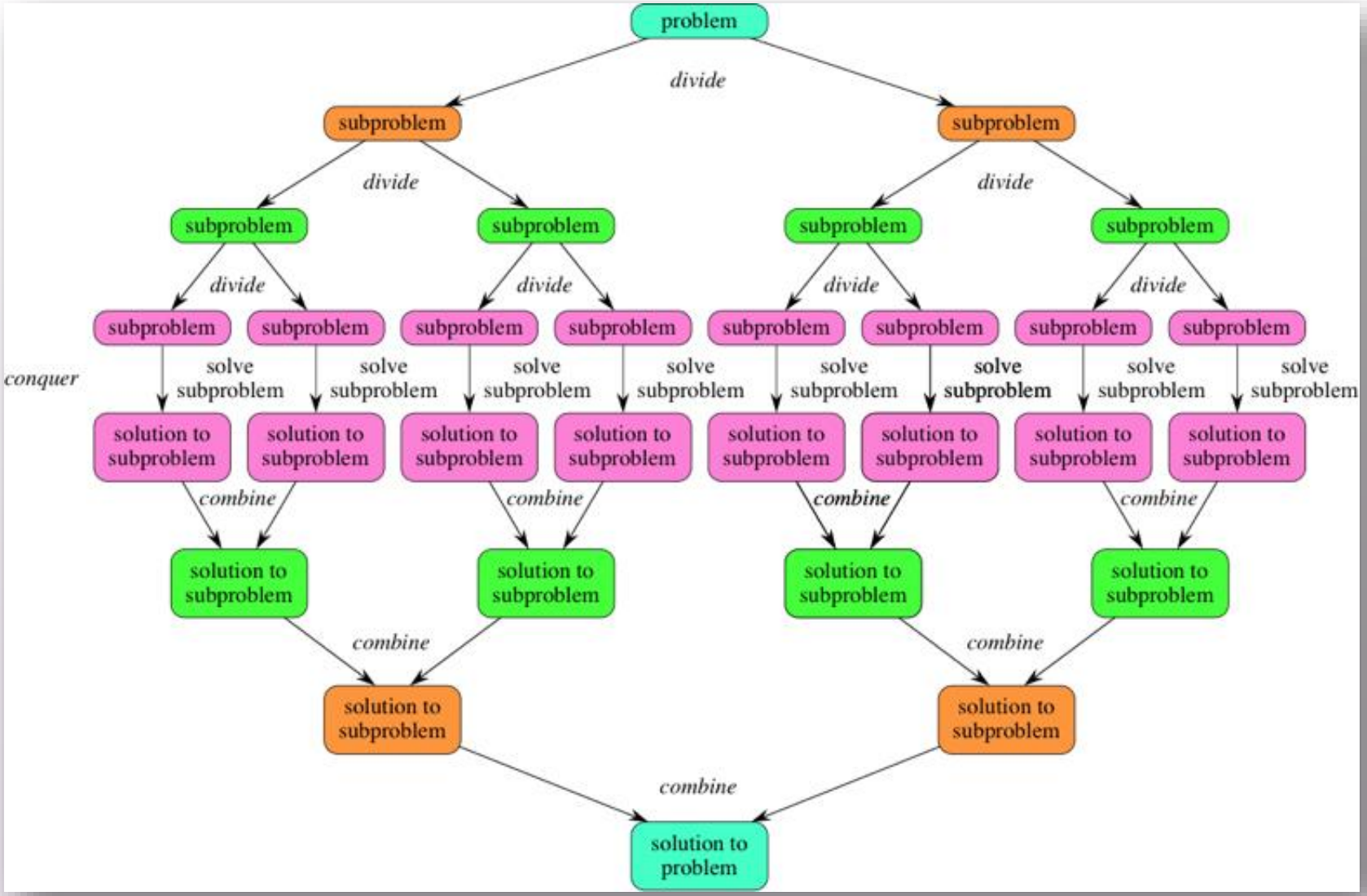
Relembrando...

- Na aula passada, estudamos uma estrutura de dados conhecida como **Dicionários**. Os principais métodos e funções associados a esse tipo de estrutura de dados que estudamos foram:
  - len()
  - update()
  - keys(), values() e items()
  - sorted()
  - pop(), popitem() e clear()

Introdução

- Em geral, problemas complexos exigem algoritmos complexos;
- É possível dividir um grande problema em problemas menores. A isso chamamos de **dividir para conquistar**;
- Cada parte menor tem um **algoritmo mais simples** o que facilita chegar à **solução do grande problema**;
- Ainda pode existir a necessidade de **dividir as partes menores em outras partes menores** a fim de obter uma solução mais simples de uma parte do problema maior.

O "dividir para conquistar"



O que é uma função?

- Em programação de computadores, uma função é um "pedaço" de código que executa alguma tarefa e retorna (ou não) um valor (em Python, é possível retornar mais de um valor);
- Uma função pode ser invocada de qualquer parte do programa. Para isso, basta informar o seu nome e, se necessário, informar os seus parâmetros;
- A utilização de funções permite ao programador:
  - Descentralizar trechos de códigos que fazem parte do programa principal
  - Utilizar a função várias vezes sem a necessidade de reescrever o seu código

Declaração de funções

- Sintaxe:

```
def nome_da_função (par1, par2, par3, ..., parN):
    instruções
```

onde:

- **nome_da_função** é o nome escolhido que, em geral, começa com um verbo
- **par1, par2, arg3, ..., parN** são os parâmetros de entrada da função
  - Uma função pode ter nenhum, um ou vários parâmetros de entrada
- **instruções** são os comandos que serão executados quando a função for invocada

Retorno de funções

- Em Python, geralmente um função computa um ou mais valores;
- Para indicar o(s) valor(es) que será(ão) devolvido(s), uma função deve usar a palavra-chave **`return`**;
- Ao encontrar **`return`**, a função devolve um (ou mais valores) e é encerrada. Em seguida, o controle passa para o ponto em que ela foi invocada;
- Se uma função for encerrada e nenhum valor de retorno for especificado, o valor retornado é **`None`**.

Exercício Resolvido 01

- Crie um arquivo chamado **MinhasFuncoes.py** e defina duas funções:

```
def fazer_saudacao1():
     return

def fazer_saudacao2():
     return "Oi"
```

- Na **mesma pasta** de MinhasFuncoes.py, crie outro arquivo chamado **TestaFuncoes.py**:

```
from MinhasFuncoes import *

print(fazer_saudacao1()) #imprime None
print(fazer_saudacao2()) #imprime Oi
```

Escopo de variáveis

- O escopo de variáveis está relacionado à forma como a variável é identificada dentro de um programa, em relação às funções que compõem esse programa;

- Uma variável pode ser do tipo **global** ou **local**:
  - **Global**
    - É declarada dentro do programa principal
    - É visível para todas as funções hierarquicamente subordinadas à rotina chamadora
  - **Local**
    - É declarada dentro de uma função e é válida somente dentro dela
    - As outras funções do programa não podem usar essas variáveis

Exercício Resolvido 02 – 1 de 2

- **No início** do arquivo **MinhasFuncoes.py**, declare uma variável chamada `uma_variável_global`;

- Em seguida, **acrescente** duas novas funções ao arquivo:

```
uma_variavel_global = "variável global"

. . .

def testa_escopo01():
    uma_variavel_local = "variável local da função 01"
    return uma_variavel_local, uma_variavel_global

def testa_escopo02():
    '''Ocorrerá um erro porque v_local é uma variável local
    definida dentro de testa_escopo01() e, portanto, não é
    visível dentro de testa_escopo02()'''
    return uma_variavel_local, uma_variavel_global
```

Exercício Resolvido 02 – 2 de 2

- **Acrescente** as novas chamadas ao arquivo **TestaFuncoes.py** e observe que será gerado um erro;

```
from MinhasFuncoes import *

.
.
.

uma_variavel_global = "variável global"
a, b = testa_escopo01()
print(a)
print(b)
c, d = testa_escopo02()
print(c)
print(d)
```

Argumentos

- Argumentos (ou parâmetros) são inicializados através dos valores informados no ponto em que a função foi invocada;
- Estabelecem uma comunicação entre um ponto chamador da função e a função invocada;
- Um argumento funciona como se fosse uma variável local, ou seja, só existe dentro da função que o declarou.

Exercício Resolvido 03 – 1 de 2

- Em MinhasFuncoes.py, adicione mais duas funções que receberão, cada uma, uma lista de números inteiros, como argumento
  - **`retorna_soma()`**: deve retornar a soma dos elementos da lista
  - **`retorna_menormaior()`**: deve retornar o menor e o maior elemento da lista

```
.
.
.

def retorna_soma(lista):
    return sum(lista)

def retorna_menormaior(lista):
    return min(lista), max(lista)
```

Exercício Resolvido 03 – 2 de 2

- Em TestaFuncoes.py, acrescente as novas chamadas:

```
from MinhasFuncoes import *

.
.
.

L = [3, 7, 1, 9, 12]
print("L = {}".format(L))

print("Soma = {}".format(retorna_soma(L)))

# menor receberá o primeiro valor retornado pela função
# maior receberá o segundo valor retornado pela função
menor, maior = retorna_menormaior(L)
print("{} é o menor da lista e {} é o maior".format(menor, maior))
```

Exercício Resolvido 04 – 1 de 2

- **Acrescente** mais uma função ao arquivo **MinhasFuncoes.py**:

```
.
.
.

def fazer_saudacao3(sexo, nome):
    if sexo.upper() == "M":
       return "Olá, sr. {}.".format(nome)
    elif sexo.upper() == "F":
       return "Olá, sra. {}.".format(nome)
```

Exercício Resolvido 04 – 2 de 2

- Em TestaFuncoes.py, faça a chamada da nova função;

- Observe que foram passados 2 valores: "M" e "Procópio"
  - Esses valores serão recebidos, na mesma sequência, pelos 2 argumentos da função
  - Assim, sexo corresponderá a "M" e nome a "Procópio"
  - Por fim, a impressão será Olá, sr. Procópio.

```
from MinhasFuncoes import *

.
.
.

print(fazer_saudacao3("M", "Procópio"))
```

```
.
.
.

def fazer_saudacao3(sexo, nome):
    if sexo.upper() == "M":
        return "Olá, sr. {}.".format(nome)
    elif sexo.upper() == "F":
        return "Olá, sra. {}.".format(nome)
```

Exercício Resolvido 05 – 1 de 2

- Vamos a outro exemplo. **Acrescente** mais uma função ao arquivo **MinhasFuncoes.py**:

```
.
.
.

def exibir_presidentes(br, eua):
    msg  = "{} é presidente do Brasil.\n"
    msg += "{} é presidente dos EUA."
    return msg.format(br, eua)
```

Exercício Resolvido 05 – 2 de 2

- Em TestaFuncoes.py, faça a chamada da nova função;
- Observe que para ela funcionar corretamente, a ordem dos parâmetros deve ser a mesma esperada pela função:

```
from MinhasFuncoes import *

. . .

print(exibir_presidentes("Bolsonaro", "Trump"))
```

- Veja o que ocorre ser invertemos a ordem dos parâmetros:

```
print(exibir_presidentes("Trump", "Bolsonaro"))
```

Argumentos *default*

- É possível atribuir valores *default* (padrão) aos argumentos de uma função
  - Se o ponto chamador da função não especificar os valores para esses argumentos, os valores *default* são usados

- Sintaxe:

```
def nome_da_funcao(arg1, arg2 = valor2, ..., argN = valor):
    instruções da função
```

- Se apenas alguns argumentos têm valores *default*, estes devem ser os últimos da lista

Exercício Resolvido 06 – 1 de 2

- **Acrescente** mais uma função ao arquivo **MinhasFuncoes.py**:

```
.
.
.

def fazer_saudacao4(saudacao, nome = "prezado(a) aluno(a)"):
    return "{}, {}.".format(saudacao, nome)
```

Exercício Resolvido 06 – 2 de 2

- Em TestaFuncoes.py, faça a chamada da nova função

```
from MinhasFuncoes import *

. . .

print(fazer_saudacao4("Bom dia", "Procópio"))
print(fazer_saudacao4("Bom dia"))
```

- Observe que em `fazer_saudacao4()`
  - **saudacao** é um argumento que não foi atribuído um valor *default* e, portanto, obrigatoriamente o ponto chamador deve informar um valor a ser recebido por ele
  - **nome** é um argumento declarado com um valor *default*. Assim, caso não seja informado um valor para ele, este assumirá o valor padrão **prezado(a) aluno(a)**
- No exemplo, observe que o primeiro print() será uma **saudação personalizada** para Procópio. Já o segundo, a **saudação será genérica** porque não foi informado um valor para o argumento **nome**.

Documentação de funções

- Ao invés de utilizar comentários para descrever o que uma função faz, é mais interessante usar *docstrings*
  - Basta colocar uma constante *string* após o cabeçalho da função
  - O acesso à documentação é feito por meio do interpretador, usando a notação **nome_da_funcao.__doc__** (**antes** e **depois** da palavra **doc**, são colocados **2** ***underlines***)
- Sintaxe:

```
def nome_da_função (arg1, arg2, arg3, ..., argN):
    "Dentro da string, escreve-se a documentação da função"
    instruções
```

Exercício Resolvido 07

- Em MinhasFuncoes.py, **modifique `fazer_saudacao4()`** adicionando a sua documentação:

```
def fazer_saudacao4(saudacao, nome = "prezado(a) aluno(a)"):
    "Esta função é um exemplo de como utilizar argumentos default"
    return "{}, {}.".format(saudacao, nome)
```

- Para testar a documentação, **modifique TestaFuncoes.py** adicionando a seguinte linha:

```
from MinhasFuncoes import *

. . .

print(fazer_saudacao4.__doc__)
```

Exercícios de Fixação

1) Construa um programa que solicite ao usuário os dados necessários para cálculo da área de um losango. Em seguida, com base nos argumentos informados, crie uma função chamada **`calcula_area_losango()`** para retornar a área da figura. Por fim, o programa deve imprimir a área calculada.

2) Construa um programa no qual o usuário informe as coordenadas de 2 pontos. Em seguida, construa a função **`calcula_distancia_pontos()`** para retornar a distância entre os pontos, cujos argumentos são as suas respectivas coordenadas X e Y. O programa que invocou a função deve exibir a distância entre os dois pontos.

3) Construa uma função chamada **`gera_matriz_aleatoria()`** que tem como argumentos o número de linhas e o número de colunas de uma matriz. Como resposta, a função deve retornar a matriz gerada cujos valores variam no intervalo [1, 10]. Ao fim, o programa chamador deve imprimir a matriz.

Principal referência

1) Esperança, Cláudio. **Python: Funções**. Disponível em: https://slideplayer.com.br/slide/45334/. Acessado em: 15 abr. 2019.