Algoritmos e Lógica de Programação

Aula 12

Modularização de Algoritmos / Funções

Prof. Dr. Dilermando Piva Jr

1º Semestre - DSM

O que é uma FUNÇÃO?

Engenharia de Software

Modularização...

Imagem: http://felipevellozo.blogspot.com/2009/03/minhas-imagens-dos-outros-6-na

Engenharia de Software

Modularização...

Técnica altamente recomendável, que consiste em dividir um programa/algoritmo maior ou principal em partes menores ou sub-rotinas (funções) tornando-o mais estruturado,

organizado e refinado.

Engenharia de Software

Modularização em Python...

→ Funções

Engenharia de Software

Modularização em Python...

→ Funções (builtin...)

- abs()
- min()
- max()
- divmod() →a,b = divmod(5,2)→ a=2,b=1
- pow() → a = pow(3,2) → a=9
- len()

Engenharia de Software

Modularização em Python...

→ Funções

Uma função é um recurso (estático) que tem como objetivo a execução de um conjunto de comandos para atingir / realizar um objetivo ESPECÍFICO.
PRINCIPALMENTE QUANDO ESSE PROCESSO É REALIZADO COM FREQUENCIA (REPETIDO VÁRIAS VEZES)
DRY – **D**on´t **R**epeat **Y**ourself
Para tanto, essa função pode ou não ter parâmetros. Ela pode ou não retornar algum valor.

Funções

Execução de um conjunto de ações/comandos para atingir um objetivo ESPECÍFICO

PASSAR MENSAGEM PARA GERENTES
REUNIÃO SEMANAL COM A DIREÇÃO.

PASSE O EMAIL PARA OS GERENTES

Funções

Execução de um conjunto de ações/comandos para atingir um objetivo ESPECÍFICO

FUNÇÃO SEM PARÂMETRO E SEM RETORNO

PASSAR MENSAGEM PARA GERENTES
REUNIÃO SEMANAL COM A DIREÇÃO.

PASSE O EMAIL PARA OS GERENTES

EXEMPLO: FUNÇÃO reset()

Funções

Execução de um conjunto de ações/comandos para atingir um objetivo ESPECÍFICO

PASSAR MENSAGEM PARA O CLIENTE ESPECIFICADO AGRADECENDO A COMPRA DOS PRODUTOS INFORMADOS.

PASSE O EMAIL PARA O CLIENTE <ABC> AGRADECENDO A COMPRA DOS PRODUTOS <X e Y>

Funções

Execução de um conjunto de ações/comandos para atingir um objetivo ESPECÍFICO

FUNÇÃO COM PARÂMETRO E SEM RETORNO

PASSAR MENSAGEM PARA O CLIENTE ESPECIFICADO AGRADECENDO A COMPRA DOS PRODUTOS INFORMADOS.

PASSE O EMAIL PARA O CLIENTE <ABC> AGRADECENDO A COMPRA DOS PRODUTOS <X e Y>

EXEMPLO: FUNÇÃO print("texto")

Funções

Execução de um conjunto de ações/comandos para atingir um objetivo ESPECÍFICO

PASSE O EMAIL PARA O CLIENTE <ABC> SOLICITANDO UM DIA E HORÁRIO PARA UMA REUNIÃO

PASSAR MENSAGEM PARA O CLIENTE ESPECIFICADO SOLICITANDO UM DIA E HORÁRIO PARA REUNIÃO

DIA e HORA

Funções

Execução de um conjunto de ações/comandos para atingir um objetivo ESPECÍFICO

FUNÇÃO COM PARÂMETRO E COM RETORNO

PASSE O EMAIL PARA O CLIENTE <ABC> SOLICITANDO UM DIA E HORÁRIO PARA UMA REUNIÃO

PASSAR MENSAGEM PARA O CLIENTE ESPECIFICADO SOLICITANDO UM DIA E HORÁRIO PARA REUNIÃO

DIA e HORA

EXEMPLO: FUNÇÃO int(3.56)

3 ←

Funções

Execução de um conjunto de ações/comandos para atingir um objetivo ESPECÍFICO

← → Bloco de comandos

(identação)

Funções

Execução de um conjunto de ações/comandos para atingir um objetivo ESPECÍFICO

def <identificador>:

← → Bloco de comandos

(identação)

Funções

Execução de um conjunto de ações/comandos para atingir um objetivo ESPECÍFICO

def <identificador> (lista parâmetros):

← → Bloco de comandos

(identação)

Funções

Execução de um conjunto de ações/comandos para atingir um objetivo ESPECÍFICO

def <identificador> (lista parâmetros):

← → Bloco de comandos

(identação) return <VALOR>

Funções

Execução de um conjunto de ações/comandos para atingir um objetivo ESPECÍFICO

Para chamar a função

<identificador> ()

<identificador> (parâmetros)

Variável = <identif.> (parâmet.)

Função sem Retorno

Funções sem Retorno

def <identificador> ():

← → Bloco de comandos

(identação)

Funções sem Retorno

def <identificador> ():

← → Bloco de comandos

(identação)

GERALMENTE UTILIZADA PARA A OTIMIZAÇÃO DE CÓDIGO REPETITIVO

Ex:

MOSTRAR CREDENCIAIS

Exemplo

```
def mostra_rodape():
    print("-----------------------------------------")
    print("| PyPRO - Seja um profissional Python!  |")
    print("-----------------------------------------")

#...

print("Execução do programa")
print("Agora vamos chamar a função para mostrar o rodape")
mostra_rodape()
print("Execução de mais alguns comandos")
print("Mostra mais uma vez o rodapé")
mostra_rodape()
print("Fim do programa")
```

Função com Retorno

Funções com Retorno

def <identificador> ():

← → Bloco de comandos

(identação) return <lista de retornos>

Exemplo

```
def um_megabits():
    valor = 1024 * 1024
    valor2 = pow(2, 20)
    return valor


# ...

x = um_megabits()
print(f"O total de bits é: {x}")
```

Função com Retorno

Funções com Retorno

def <identificador> ():

← → Bloco de comandos

(identação) return <lista de retornos>

Exemplo 1

```
def area_circulo(raio):
    PI = 3.141592
    area = PI * pow(raio, 2)
    return area

#...

r = float(input("Digite o raio: "))
print(f"A área do circulo de raio {r} é igual a {area_circulo(r)}")
```

Exemplo 2

```
#Funções que chamam outras funções

def area_circulo(raio):
    PI = 3.141592
    area = PI * pow(raio, 2)
    return area

def area_cilindro(altura, raio):
    area = area_circulo(raio)*altura
    return area

#...

r = float(input("Digite o raio: "))
h = float(input("Digite a altura: "))
print(f"A área do cilindro de raio {r} e altura {h} é igual a {area_cilindro(r,h)}")
```

Função com Parâmetro Padrão

Funções com Parâmetro Padrão

def <identificador> **(<id>=valor**, ...):

← → Bloco de comandos

(identação) return <lista de retornos>

Quando um parâmetro pode ser omitido!!

Funções com Parâmetro Padrão

Veja essa situação...

Vamos criar uma função chamada potencia.

Você pode passar um ou dois parâmetros.

Se você passar apenas um parâmetro, ela calculará o quadrado de um número (potencia 2). Caso passe dois parâmetros, o segundo será utilizado como o valor da potencia.

Ex: potencia(3) → $3^2 = 9$

potencia(3,4) → $3^4 = 3x3x3x3 = 81$

Parâmetros nomeados...

```
Nome_completo(nome, sobrenome):
    print(nome, sobrenome)
```

O que aconteceria se invertêssemos a ordem de entrada dos parâmetros?

Exemplo

```
#Funções com parâmetro padrão

def potencia(numero, expoente = 2):
    resultado = pow(numero, expoente)
    return resultado

#...

n = float(input("Digite o numero: "))
e = int(input("Expoente: "))

print(f"Valor com expoente: {potencia(n,e)}")
print(f"Valor sem o expoente: {potencia(n)}")
```

Documentando Funções com DocStrings

Funções com Parâmetro Padrão

def <identificador> (<id>:valor, ...):

← → ‘‘‘ **docstring** ‘‘‘

← → Bloco de comandos

(identação) return <lista de retornos>

Exemplo

```
#Documentando uma função com docstring

def potencia(numero, expoente = 2):
    """ Função que calcula a potencia de um número.
Valor de Entrada:
numero = numero a ser calculado (elevado a potencia) (float)
expoente = expoente utilizado no cálculo (inteiro)
Resultado:
Resultado do cálculo (float)"""

    resultado = pow(numero, expoente)
    return resultado

#...

n = float(input("Digite o numero: "))
e = int(input("Expoente: "))

print(f"Valor com expoente: {potencia(n,e)}")
print(f"Valor sem o expoente: {potencia(n)}")
print()
print()
help(potencia)
```

Parâmetros
*args e **kwargs

Parâmetros *args e **kwargs

def <identificador> **(*args **kwargs):**

← → Bloco de comandos

(identação) return <lista de retornos>

UTILIZAMOS *ARGS E *KWARGS QUANDO OS PARÂMETROS SÃO VARIÁVEIS.

Parâmetro *args

```
def soma(n1, n2, n3):
    total = n1+n2+n3
    return total
```

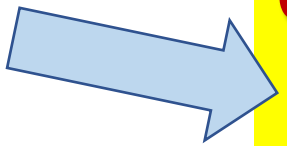
Parâmetro *args

```
def soma(n1, n2, n3):
    total = n1+n2+n3
    return total
```

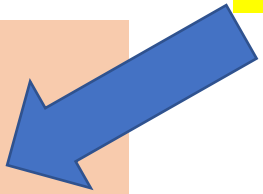
```
def soma(*args):
    print(args)
```

Parâmetro *args

```
def soma(n1, n2, n3):
    total = n1+n2+n3
    return total
```



```
def soma(*args):
    print(args)
```



```
def soma(*args):
    total = 0
    for i in args:
        total += i
    return total
```

```
def soma(*args):
    return sum(args)
```

Parâmetro *args

```
def soma(n1, n2, n3):
    total = n1+n2+n3
    return total
```

```
def soma(*args):
    print(args)
```

```
def soma(*args):
    total = 0
    for i in args:
        total += i
    return total
```

```
def soma(*args):
    return sum(args)
```

Parâmetro **kwargs

```
def comida_favorita(**kwargs):
    print(kwargs)
```

➔ Recebe em um dicionário!!

Parâmetros... ORDEM

1) PARÂMETROS DEFINIDOS
2) *args
3) PARÂMETROS NOMEADOS
4) **kwargs

Exemplo 1

```
#SEM Padrâmetro *args

def soma_numeros(n1, n2, n3):
    return (n1+n2+n3)

print(soma_numeros(1,2,3))

print(soma_numeros(1,2))

print(soma_numeros(1,2,3,4,5,6))
```

```
#COM Padrâmetro *args

def soma_numeros(*args):
    print(args)

soma_numeros(1,2,3)

soma_numeros(1,2)

soma_numeros(1,2,3,4,5,6)
```

Exemplo 2

```
#Padrâmetro **kwargs

def cores_favoritas(**kwargs):
    for nome in kwargs:
        print(f"{nome} gosta da cor {kwargs[nome]}")

print("1----------")
cores_favoritas(ana="vermelho", marcelo="amarelo")
print("2----------")
cores_favoritas(ana="vermelho", marcelo="amarelo", fernanda="branco")
```

VAMOS PARA A PRÁTICA ?!!!

EXERCÍCIO 1

Faça uma função que retorne o valor lógico V (verdadeiro) se o número inteiro passado por parâmetro for par, e F (falso) se não.

Implemente sua função em um programa completo.

EXERCÍCIO 2

Faça uma função que retorne o valor lógico V (verdadeiro) se o número inteiro passado por parâmetro for primo, e F (falso) se não.

Implemente sua função em um programa completo.

EXERCÍCIO 3

Faça uma função que determine se um ano qualquer, no formato AAAA, é bissexto.
A função retorna 1 se o ano é bissexto e 0(zero) se não.

Divisível por 4
Se for divisível por 100, tem que ser também divisível por 400

EXERCÍCIO 4

Construa uma função que retorne o MDC de dois números inteiros passados por parâmetro.

EXERCÍCIO 5

Faça uma função que receba como parâmetro o raio de uma esfera, calcule e retorne o valor de seu volume.

Volume da Esfera : $v = 4/3 \pi * R^3$

EXERCÍCIO 6

Crie uma função que receba como parâmetro 3 números interios (representando horas, minutos e segundos). A função deve converter em segundos.

Por exemplo: 2 h, 40 min e 10 segundos correspondem a 9.610 segundos.

EXERCÍCIO 7

Crie uma função que recebe como parâmetro um número inteiro positivo, maior que zero.
Esta função mostra (imprime) a quantidade de números primos que existe entre 1 e n

Se n = 3, ela imprimiria o valor 3, pois existem 3 números primos (1,2,3)
Se n = 10, ela imprimiria o valor 5, pois existem 5 números pirmos (1, 2, 3, 5, 7)