INSTITUTO FEDERAL
São Paulo
Câmpus São Carlos

Módulos e Recursão
AP1S1 – Algoritmos e Programação

Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas

INSTITUTO FEDERAL
São Paulo
Câmpus São Carlos

Módulos

- Um **módulo** é um arquivo contendo definições e comandos em Python para serem usados em outros programas em Python

- Há diversos módulos do Python que fazem parte da **biblioteca padrão**

- Uma vez que o módulo é importado, podemos utilizar as coisas que estão definidas dentro dele

Módulos – Como encontrá-los?

- A documentação para a versão 3 do Python está disponível em http://docs.python.org/py3k/
- Essa é uma referência muito útil sobre todos os aspectos do Python
- Contém uma listagem de todos os módulos padrões disponíveis no Python (veja Global Module Index)
- Existe também um manual de referência à linguagem (Language Reference) e um tutorial (Tutorial), bem como instruções para instalação, dicas de como fazer, e respostas a perguntas frequentes
- Use essas informações com frequência!!!!

Módulos – Como importá-los?

- Para importar deve-se utilizar o comando **import**:

- import package

```
import math
print(math.sqrt(25))
```

- from package import item

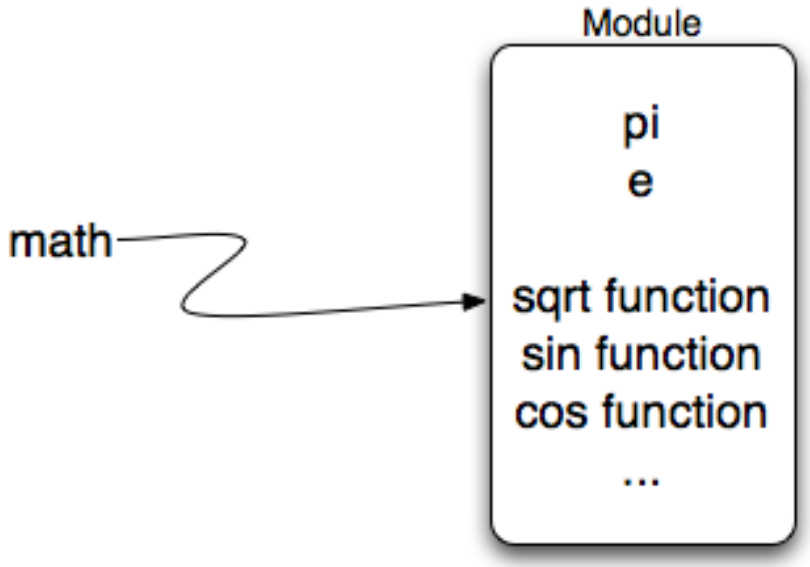
```
from math import sqrt
print(sqrt(25))
```

- Ao utilizar “from package import item”, o item pode ser um subpacote, submódulo, classe, função ou variável.

Módulos – Módulo *math*

- O módulo math contém funções matemáticas que se costuma encontrar em calculadoras e algumas constantes matemáticas como *pi* e *e*

- Ao importar o módulo *math*, obtém-se uma referência para um objeto module que contém esses elementos



Módulos – Módulo *math*

TesteMath.py - C:/Users/silvana/Documents/IFSP/ALG1/Scripts/TesteMath.py (3.6.3)

File Edit Format Run Options Window Help

```
import math

print(math.pi)
print(math.e)

print(math.sqrt(2.0))

print(math.sin(math.radians(90)))    # seno de 90 graus
```

Python 3.6.3 Shell

File Edit Shell Debug Options Window Help

```
Python 3.6.3 (v3.6.3:2c5fed8, Oct  3 2017, 17:26:49) [MSC v.1900 32 bit (Intel)]
 on win32
Type "copyright", "credits" or "license()" for more information.
>>>
===== RESTART: C:/Users/silvana/Documents/IFSP/ALG1/Scripts/TesteMath.py =====
3.141592653589793
2.718281828459045
1.4142135623730951
1.0
>>>
```

Módulos – Módulo *random*

TesteRandom.py - C:/Users/silvana/Documents/IFSP/ALG1/Scripts/TesteRandom.py (3.6.3)

File Edit Format Run Options Window Help

```
import random

prob = random.random()
print(prob)

lanceDado = random.randrange(1,7)   # retorna um int, dentre 1,2,3,4,5,6
print(lanceDado)
```

Python 3.6.3 Shell

File Edit Shell Debug Options Window Help

```
Python 3.6.3 (v3.6.3:2c5fed8, Oct  3 2017, 17:26:49) [MSC v.1900 32 bit (Intel)]
 on win32
Type "copyright", "credits" or "license()" for more information.
>>>
==== RESTART: C:/Users/silvana/Documents/IFSP/ALG1/Scripts/TesteRandom.py ====
0.10040302555013436
3
>>>
```

Módulos - Exemplo

Arquivo ValidaValores.py → MÓDULO

```
import string

def ValidaInteiroPositivo(nro):
    for i in nro:
        if i not in string.digits:
            return False
    return True
```

Módulos - Exemplo

Arquivo qualquer .py

```
import ValidaValores
if ValidaValores.ValidaInteiroPositivo(input("Digite um valor: ")):
 print("O Número digitado é um Inteiro Positivo!")
else:
    print("O Número digitado não é um Inteiro Positivo!")
```

→Via Moodle abra os arquivos: ValidaValores.py e TestedeConversão.py

Recursão

- **Recursão** é um método de resolução de problemas que envolve quebrar um problema em subproblemas menores e menores até chegar a um problema pequeno o suficiente para que ele possa ser resolvido trivialmente

- Normalmente recursão envolve uma função que chama a si mesma

- Embora não seja fácil perceber, a recursão nos permite escrever soluções elegantes para problemas que, de outra forma, podem ser muito difíceis de programar

Recursão – Entendendo com exemplo

- Suponha que você deseja calcular a soma de uma lista de números, tais como: [1,3,5,7,9]

- Implementação conhecida:

```
def somalista(lista):
    soma = 0
    for i in lista:
        soma = soma + i
    return soma

print(somalista([1,3,5,7,9]))
```

Recursão – Entendendo com exemplo

- Como você calcularia a soma de uma lista de números se não tivesse os laços while ou for?
- Se você fosse um matemático poderia começar recordando que a adição é uma função definida para dois parâmetros, um par de números
- Para redefinir o problema da adição de uma lista para a adição de pares de números, podemos reescrever a lista como uma expressão totalmente entre parênteses. Tal expressão poderia ser algo como:

(((((1+3)+5)+7)+9)

- Poderíamos também colocar os parênteses na ordem reversa,

(1+(3+(5+(7+9))))

Recursão – Entendendo com exemplo

- Como podemos usar essa ideia e transformá-la em um programa Python?

- Em primeiro lugar, vamos reformular o problema soma em termos de listas de Python

- Poderíamos dizer que a soma da lista “lista” é a soma do primeiro elemento da lista (lista [0]), com a soma dos números no resto da lista (lista [1:])

- De forma funcional podemos escrever:

somalista(lista)=primeiro(lista)+somalista(resto(lista))

Recursão – Entendendo com exemplo

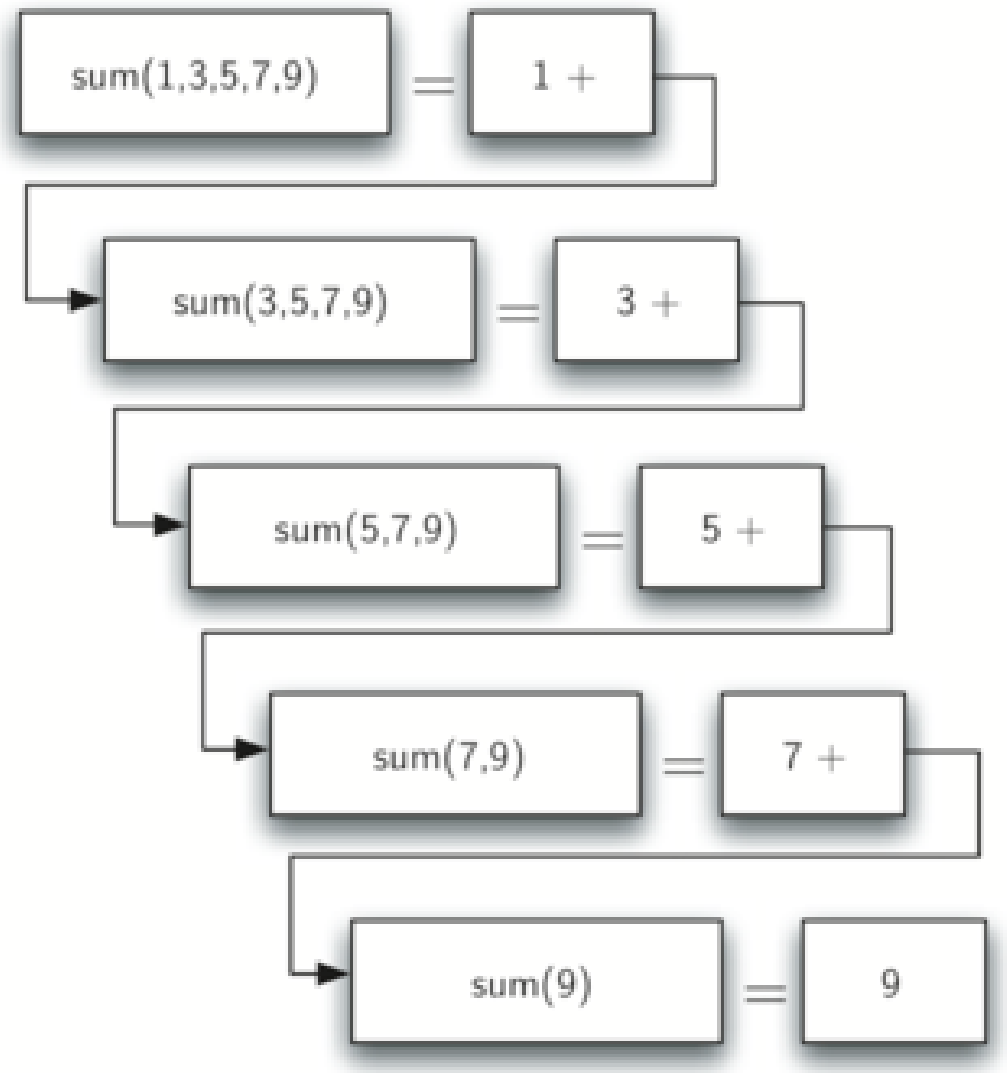
- A implementação ficaria assim:

```
def sum(lista):
    if len(lista) == 1:
        return lista[0]
    else:
        return lista[0] + sum(lista[1:])

print(sum([1,3,5,7,9]))
```

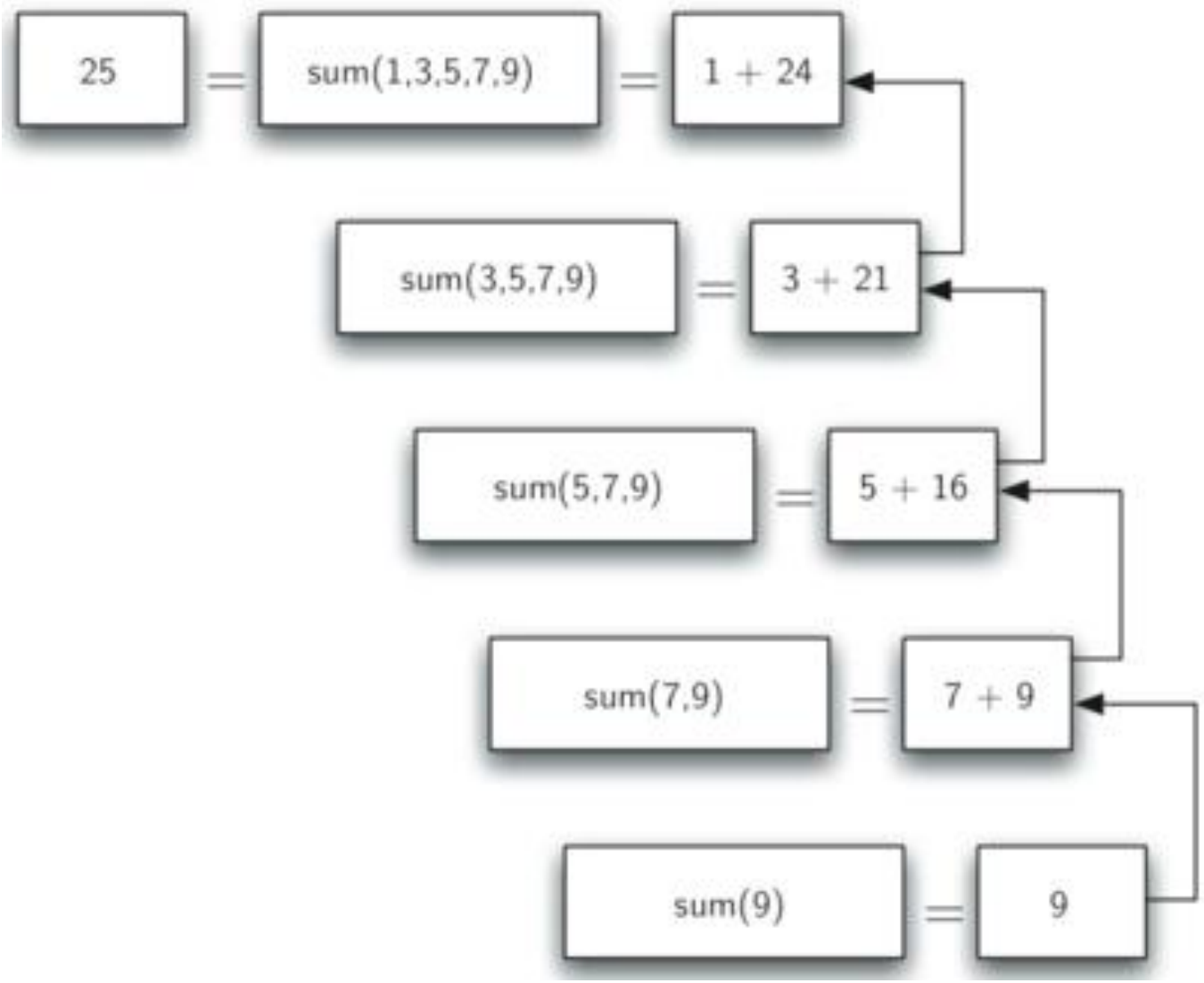
Recursão – Entendendo com exemplo

- A execução ficaria assim:



Recursão – Entendendo com exemplo

- Retornos:



Recursão – As três leis

- Todos os algoritmos recursivos devem obedecer a três leis importantes:
  - Um algoritmo recursivo deve ter um *caso básico*
  - Um algoritmo recursivo deve mudar o seu estado e se aproximar do caso básico
  - Um algoritmo recursivo deve chamar a si mesmo, recursivamente

Recursão – As três leis

- Exemplo: Encontre a definição recursiva de:

1. x * y
2. 3 * 4 = 3 + (3*3)
3. 3 * 3 = 3 + (3*2)
4. 3 * 2 = 3 + (3*1)
5. 3 * 1 = 3 + (3*0)
6. 3 * 0 = 0

Caso Trivial? x*0 = 0

Passo Recursivo? x* y = x + x*(y-1)

$$x * y = \begin{cases} 0, y = 0 \\ x + x * (y-1), y > 0 \end{cases}$$

Exemplo – Sequência de Fibonacci

- Sequência de Fibonacci: 0, 1, 1, 2, 3, 5, 8, 13, 21, 34...

n	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
F(n)	0	1	1	2	3	5	8	13	21	34	55	89	144	233	377	610	987	1597	2584	4181	6765	10946

- Os elementos iniciais são: F(0)=0 e F(1)=1

- Os demais elementos são definidos em função da soma dos dois elementos imediatamente anteriores

Exemplo – Sequência de Fibonacci

```
def termo_fibonacci(n):
    if n <= 1:
        return n
    else:
        return termo_fibonacci(n-1) + termo_fibonacci(n-2)



nro=int(input('Digite o termo da sequência de Fibonacci: '))
if nro < 0:
    print("O número deve ser maior ou igual a zero!")
else:
    print(termo_fibonacci(nro))
```

Exemplo – Sequência de Fibonacci

```
def sequencia_fibonacci(a, b, n):
    if a < n:
        print(a," ",end="")
    if b < n:
        sequencia_fibonacci(b, a + b, n)


nro=int(input('Digite o valor de parada da sequência de Fibonacci: '))
sequencia_fibonacci(0,1,nro)
```

Exercícios de Recursão

1. Escreva um algoritmo recursivo para calcular o fatorial de um número
2. Escreva um algoritmo recursivo para calcular x^y
3. Escreva um algoritmo recursivo para calcular o quociente da divisão inteira
4. Escreva um algoritmo recursivo para calcular o resto da divisão inteira
5. Escreva um algoritmo recursivo para retornar a quantidade de caracteres de uma string
6. Crie uma função recursiva que exiba verticalmente uma string

Exercícios

7. Crie uma função recursiva para exibir os algarismos de um número positivo de forma invertida. Exemplo: 326

```
6
2
3
```

8. Crie uma função recursiva para exibir os algarismos de um número positivo na ordem correta. Exemplo: 326

```
3
2
6
```

9. Escreva um algoritmo para gerar a sequencia de Fibonacci até que o valor gerado atinja um valor limite n. Considere que a sequência começa em 1 (1º termo).

Exercícios de Módulo

1. Crie um módulo de verificação de valores de entrada como:
    a) Número inteiro positivo
    b) Número inteiro – positivos e negativos
    c) Números reais – positivos e negativos

Esse módulo deverá ser importado e suas funções utilizadas para validações de entrada de outras funções ou scripts.

2. Crie um módulo para conter funções importantes para a manipulação de listas, considerando 2 listas como parâmetros de entrada: união, intersecção, diferença, soma, subtração, multiplicação, divisão, etc, retornando a matriz resultado.

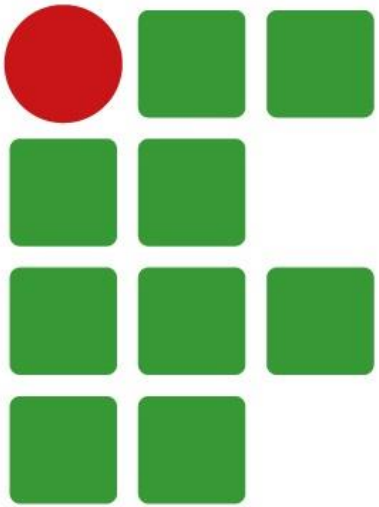
3. Crie um módulo para conter funções para a manipulação de matrizes, considerando 2 matrizes como entrada: soma, subtração, produto de matrizes, etc, retornando a matriz resultado.

Exercícios

4. Considere o cenário de uma Escola. Faça um programa modularizado para a escola gerenciar o cadastramento de dados dos alunos. O menu principal do sistema que deverá ficar aparecendo continuamente na tela é o seguinte:

    1. **Cadastrar Aluno**
    2. **Atualizar Aluno**
    3. **Excluir Aluno**
    4. **Listar Dados de Aluno**
    5. **Listar todos os Alunos**
    6. **Sair**

- Ao escolher a opção “6” o sistema deverá ser finalizado. Os dados dos alunos que necessitam serem registrados são: RA, Nome, Data de Nascimento, Sexo, Telefone e E-mail. Cada Aluno possui um RA único, que não se repete entre os outros alunos. Desse modo, ao cadastrar um novo aluno deve-se verificar se o RA já existe, e se existir, mostrar uma mensagem de erro condizente na tela. Já para as opções “2” e “3” é necessário que o aluno exista no sistema para que a operação seja realizada.
- Pense na melhor estrutura de dados para armazenar os dados no sistema. Crie funções modularizadas para cada item. Crie um arquivo OperaçõesAluno.py para conter as funções de 1 a 5. O menu deve ser o programa principal e estar em outro arquivo.



INSTITUTO FEDERAL

São Paulo

Câmpus São Carlos