Funções

CIÊNCIA DA COMPUTAÇÃO

ATITUS EDUCAÇÃO

- Aula 13 -
Pensamento Computacional

Prof. Me. Lucas R. C. Pessutto

ATITUS EDUCAÇÃO

Na aula de hoje...

01 **Subprogramação**

Motivação
Conceito
Níveis de Chamada

02 **Funções**

Sintaxe
Parâmetros e Retorno
Tipos de Funções

03 **Funções com Parâmetros**

04 **Funções com Retorno**

05 **Modularização**

Conceito
Definindo módulos personalizados

ALTUS EDUCAÇÃO

Motivação Inicial

Algoritmo **Motivacao**

1. Início
2. Preencher a lista A(de tamanho MAX) com valores do índice.
3. Imprimir a lista A
4. Somar 2 em cada elemento de A
5. Imprimir a lista A
6. Zerar os valores pares de A
7. Imprimir a lista A
8. Fim

Motivação Inicial

```python
MAX = 10
a = []
# Preencher a lista com o valor do índice
for i in range(MAX):
    a.append(i)
# Imprimindo a lista a
for i in range(MAX):
    print(a[i], end="")
    if i < MAX - 1:
        print(" - ", end="")
print()
# Somando dois a cada elemento de a
for i in range(MAX):
    a[i] += 2
```

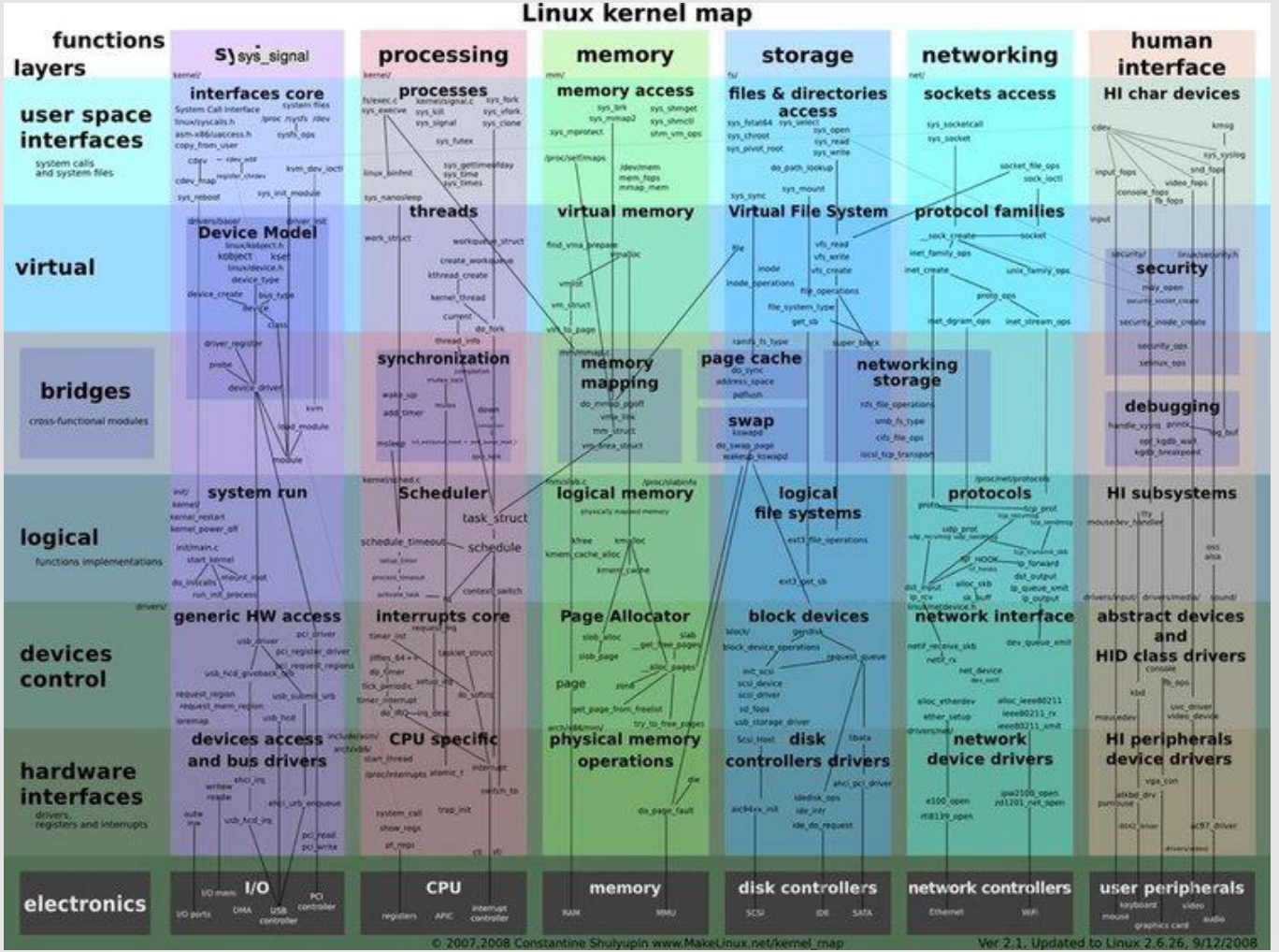
```python
# Imprimindo a lista a
for i in range(MAX):
    print(a[i], end="")
    if i < MAX - 1:
        print(" - ", end="")
print()
# Zerando os valores pares de A
for i in range(0, MAX, 2):
    a[i] = 0
# Imprimindo a lista a
for i in range(MAX):
    print(a[i], end="")
    if i < MAX - 1:
        print(" - ", end="")
print()
```

Qual o problema deste código?

Linux 3.6 ~ 15.9 milhões de linhas de código

Linux 4.2 ~ 20.2 milhões de linhas de código

Linux 5.6 ~ 33 milhões de linhas de código

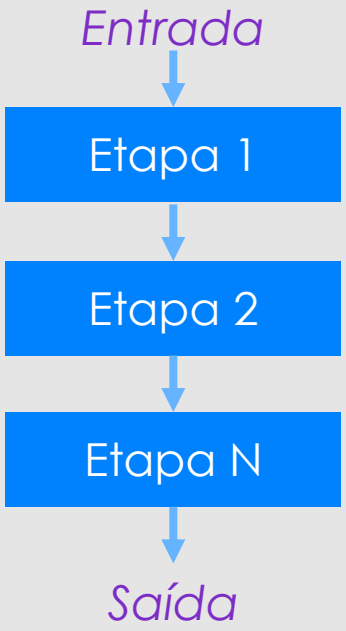


https://www.informationisbeautiful.net/visualizations/million-lines-of-code/

Programação Estruturada

Características:

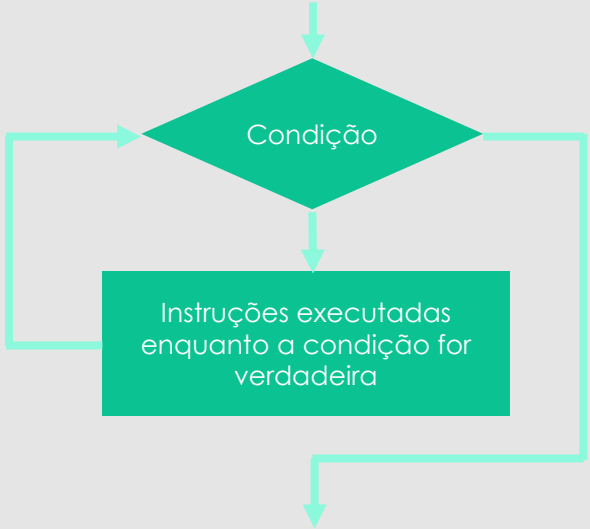
* Desenvolvimento de algoritmos por etapas



Programação Estruturada

Características:

* Desenvolvimento de algoritmos por etapas
* Número Limitado de Estruturas de Controle



Programação Estruturada

Características:

* Desenvolvimento de algoritmos por etapas
* Número Limitado de Estruturas de Controle
* **Decomposição do algoritmo completo em módulos desenvolvidos usando subprogramas**

Dividir para Conquistar

Programação Estruturada

VANTAGENS:

* Facilidade de Debugar

* Reutilização de Código

* Legibilidade

* Confiança

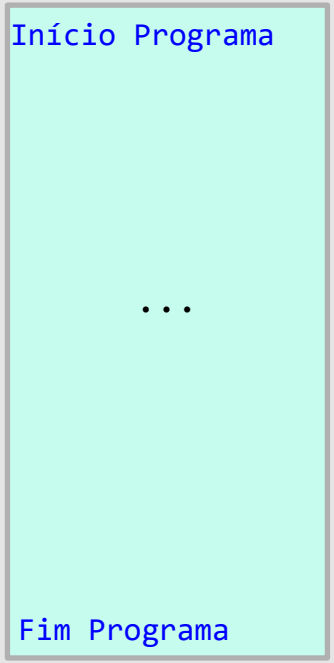
Prog. Estruturada c/ módulos e funções

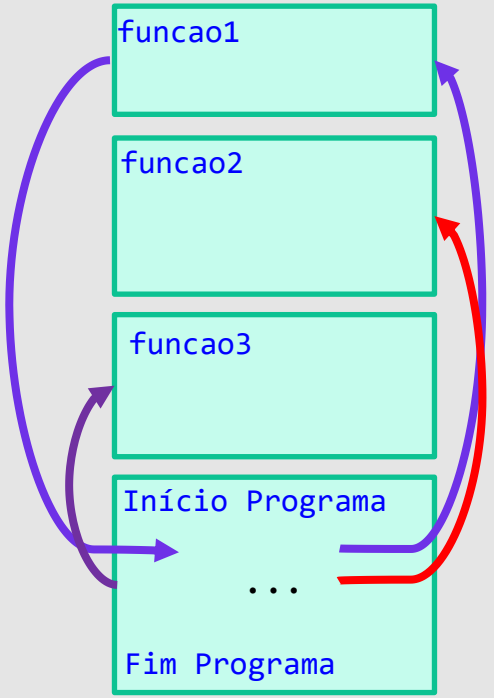
* Desenvolvimento de algoritmos por fases ou refinamentos sucessivos

* Uso de um número muito limitado de estruturas de controle

* Decomposição do algoritmo total em módulos, desenvolvidos e implementados usando subprogramas.

Funções - Ideia Principal





Motivação Inicial

```python
MAX = 10

a = []

def imprime_lista():
    # Imprimindo a lista a
    for i in range(MAX):
        print(a[i], end="")
        if i < MAX - 1:
            print(" - ", end="")
    print()
```

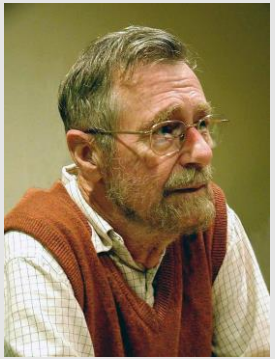
```python
# Preencher a lista com o valor do índice
for i in range(MAX):
    a.append(i)

imprime_lista()

# Somando dois a cada elemento de a
for i in range(MAX):
    a[i] += 2

imprime_lista()

# Zerando os valores pares de A
for i in range(0, MAX, 2):
    a[i] = 0

imprime_lista()
```

Subprogramação

Objetivos Principais:

* Evitar repetição de sequência de comandos

* Dividir e estruturar um programa em partes fechadas e logicamente coerentes



"A arte de programar consiste em organizar e dominar a complexidade dos sistemas"
(Dijkstra, 1972)

Funções

* Sequência de instruções ou programas executados somente quando chamados por um programa em execução, incluindo aqui o sistema operacional.

* Existem funções pré-definidas (disponibilizadas pelos compiladores) e funções desenvolvidas pelo usuário (um programa pode incluir diversas funções).

* Uma função deve executar UMA tarefa específica, muito bem identificada (programação estruturada).

* Após sua execução, o fluxo do programa retorna ao ponto imediatamente após o da chamada da função (ou subprograma).

Porque usar Funções?

* Estruturar um programa em partes logicamente coerentes

* Evita a repetição de sequências de comandos

* Copiar código é fácil e rápido, mas tende a produzir e propagar erros e dificulta a manutenção de código

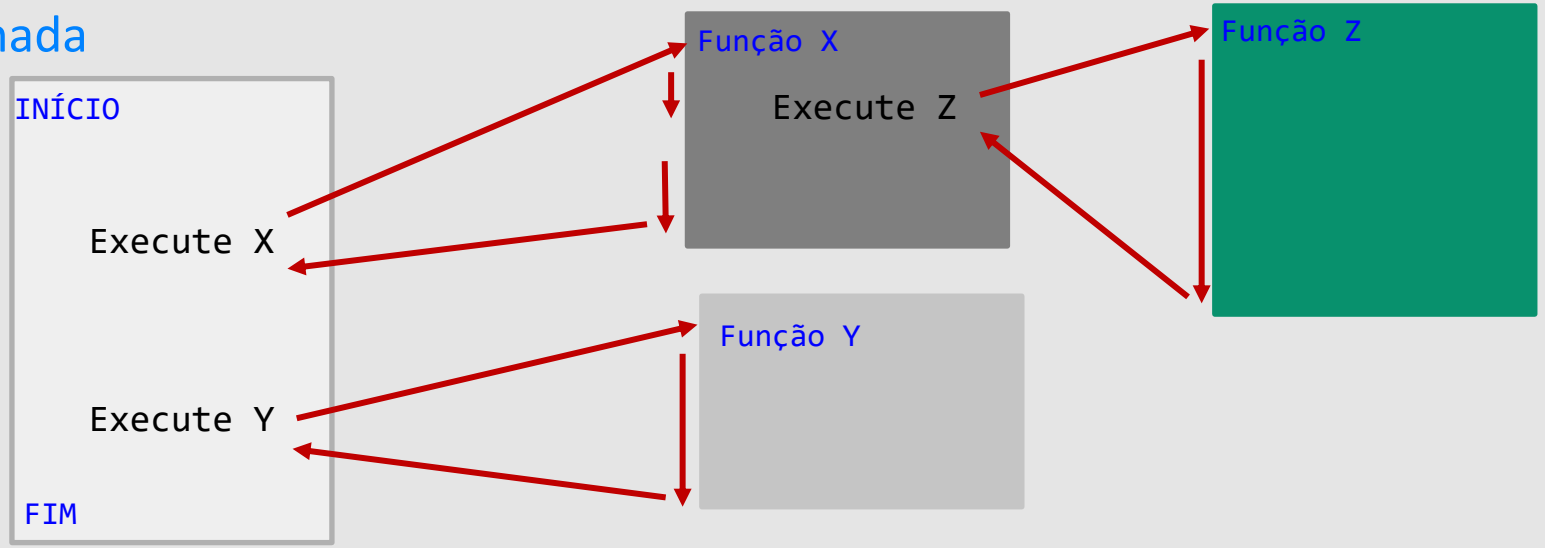
* Permite reuso de trechos de programas

* Facilita depuração e teste (funções podem ser testadas separadamente)

Funções - Níveis de Chamada

* Quando uma função é chamada, o fluxo de execução do programa se transfere para o escopo da função

* Após sua execução, o fluxo retorna ao ponto imediatamente após a chamada



Funções Pré-definidas

Já estamos acostumados a utilizar funções...

```python
import math

circulo = input("Forneça o identificador do círculo: ")
circulo = circulo.upper()

raio = float(input("Forneça o raio do círculo: "))
area = math.pi * math.pow(raio, 2)
print(f"Área do círculo {circulo} de raio {raio:.2f} é {area:.2f}")
```

Funções Pré-definidas

Funções embutidas (*built-in*) são disponibilizadas juntamente com o interpretador e podem ser usadas em **qualquer ponto** de um programa

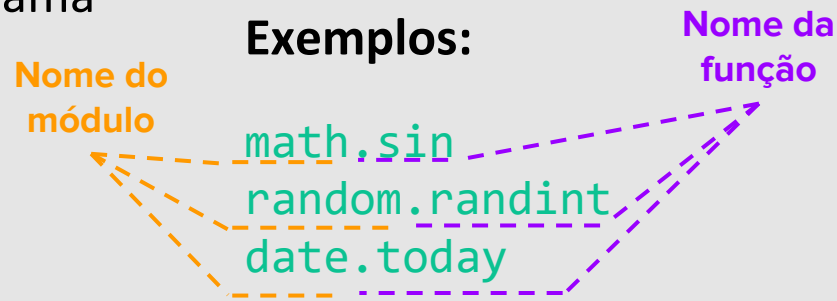
Exemplos:

`print`, `input`, `len`, `sorted`, ...

Documentação completa em
docs.python.org/3/library/functions.html

Funções de módulos podem ser **incluídas** no programa principal pelo comando `import`

Exemplos:



Lista de módulos disponíveis por padrão (muitos outros podem ser instalados)
docs.python.org/3/py-modindex.html

Funções desenvolvidas pelo programador

* Estrutura lógica mais clara

* Facilita a solução de problemas complexos através de uma solução em partes (dividir para conquistar)

* Maior facilidade de depuração e teste (subprogramas podem ser testados separadamente)

* Disponibiliza trechos de programa que solucionam problemas recorrentes

* Evita repetição de sequência de comandos (e erros decorrentes)

Declarando uma função

* Toda função precisa ser declarada para ser chamada por um programa.

* A declaração de uma função possui dois componentes:

→ **Cabeçalho**: Definição do nome, tipo de retorno e parâmetros formais

→ **Corpo**: Código da função

Declarando uma função

```
def nome (argumentos):
```

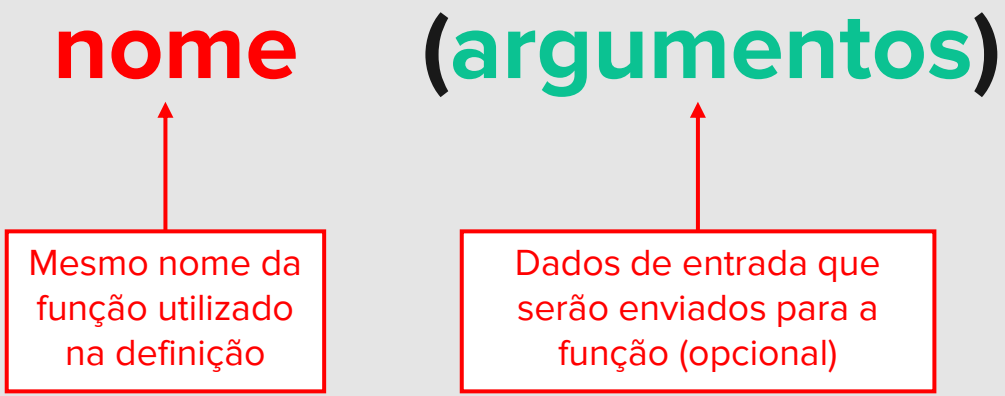
- `def`: Comando que identifica a definição de uma nova função
- `nome`: Nome da função escolhido pelo programador
- `(argumentos)`: Dados de entrada recebidos pela função (opcional, veremos isso depois)

ATITUS EDUCAÇÃO

Chamando uma função

* A chamada de uma função é feita através do seu nome

* A chamada pode ser feita em diversos pontos do programa principal após a sua definição ou a partir de outras funções



Exemplo 1

```
TAMANHO = 20

print("+", end="")
for i in range(TAMANHO):
    print("-", end="")
print("+")
print(" Números entre 1 e 5")
print("+", end="")
for i in range(TAMANHO):
    print("-", end="")
print("+")
for i in range(1, 6):
    print(" ", i)
print("+", end="")
for i in range(TAMANHO):
    print("-", end="")
print("+")
```

```
+--------------------+
 Números entre 1 e 5
+--------------------+
  1
  2
  3
  4
  5
+--------------------+
```

Quais partes deste código estão repetidas?

Exemplo 1

```
TAMANHO = 20

def imprime_cabecalho():
    print("+", end="")
    for i in range(TAMANHO):
        print("-", end="")
    print("+")
```

Declaração da Função

```
imprime_cabecalho()

print(" Números entre 1 e 5")

imprime_cabecalho()

for i in range(1, 6):
    print(" ", i)

imprime_cabecalho()
```

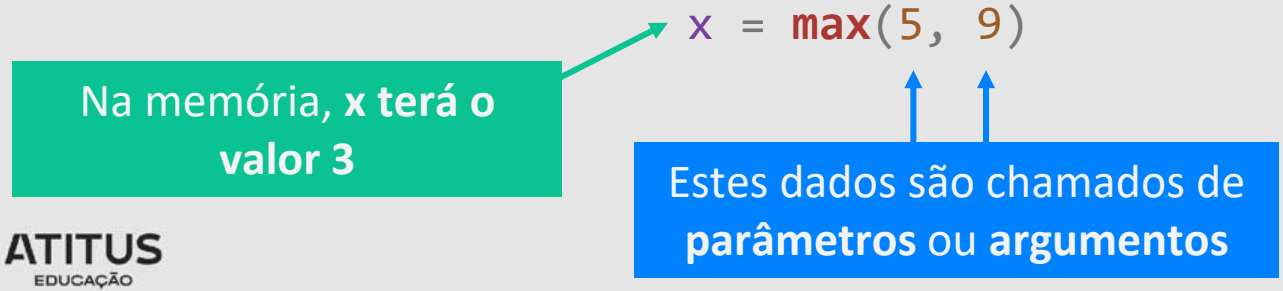
Chamada da Função

Parâmetros e Retorno

Quando um programa executa uma função:

* A função pode receber dados deste programa, utilizados localmente para executar os comandos incluídos na função: estes dados são chamados de parâmetros ou argumentos.

* Uma função pode também devolver um valor para o programa, o que chamamos de retorno de função.



Funções desenvolvidas pelo programador

	Sem Argumentos	Com Argumentos
Sem Retorno	Não recebem argumentos. Não retornam valores.	Recebem um ou mais argumentos. Não retornam valores.
Com Retorno	Não recebem argumentos. Retornam um valor.	Recebem um ou mais argumentos. Retornam um valor.

Funções sem argumento e sem retorno

Definição de função

Nome da função

Argumentos

```
def imprime_cabecalho():
    print("+", end="")
    for i in range(TAMANHO):
        print("-", end="")
    print("+")
```

Escopo da função

Funções sem argumento e sem retorno

```
imprime_cabecalho()
```
→ Chamada da função

```
print(" Números entre 1 e 5")
```

```
imprime_cabecalho()
```
→ Chamada da função

```
for i in range(1, 6):
    print(" ", i)
```

```
imprime_cabecalho()
```
→ Chamada da função

Problema 1: Menu de Opções

Escreva uma função void, de nome menu_de_opcoes, que gere o menu abaixo, sem incluir a leitura da opção informada:

```
----------
   MENU
----------

  1 - Soma de dois valores reais
  2 - Divisores do número
  3 - Sequência de números pares
  4 - Verifica se o número é perfeito

Informe a opção desejada:
        ?
```

No programa principal, inclua a chamada desta função e faça a leitura da opção desejada.

Problema 1: Menu de Opções

```
def menu_de_opcoes():
    print("-" * 10)
    print("MENU".center(10))
    print("-" * 10)
    print()
    print("  1 - Soma de dois valores reais")
    print("  2 - Divisores do número")
    print("  3 - Sequência de números pares ")
    print("  4 - Verifica se o número é perfeito")

    print("\nInforme a opção desejada:")


menu_de_opcoes()
opcao = int(input("\t? "))
```

Não seria melhor fazer essa leitura dentro da função menu_de_opcoes?

Funções com Parâmetros

* São utilizados para “passar dados” para uma função.

* Utiliza valores para parametrizar as ações de uma função

* Permite que a função exiba comportamentos diferentes a cada chamada, dependendo dos dados informados como parâmetro

* Permite a troca de informações entre a função chamada e a função chamadora

* Habilita a construção de funções mais genéricas

Funções com Parâmetros

* Exemplos:
  - → Em funções matemáticas especificamos um valor de x para calcular um valor de y
    - Cada chamada com um x retorna um y diferente
  - → Uma função que calcula a área de um círculo precisamos especificar o comprimento do raio
    - Cada chamada com um raio retorna uma área diferente

Funções com Parâmetros

```
def nome(arg1, arg2, …, argN):
    <comando>
    <comando>
    <comando>
```

Comando que identifica a definição de uma nova função

Nome da função escolhido pelo programador

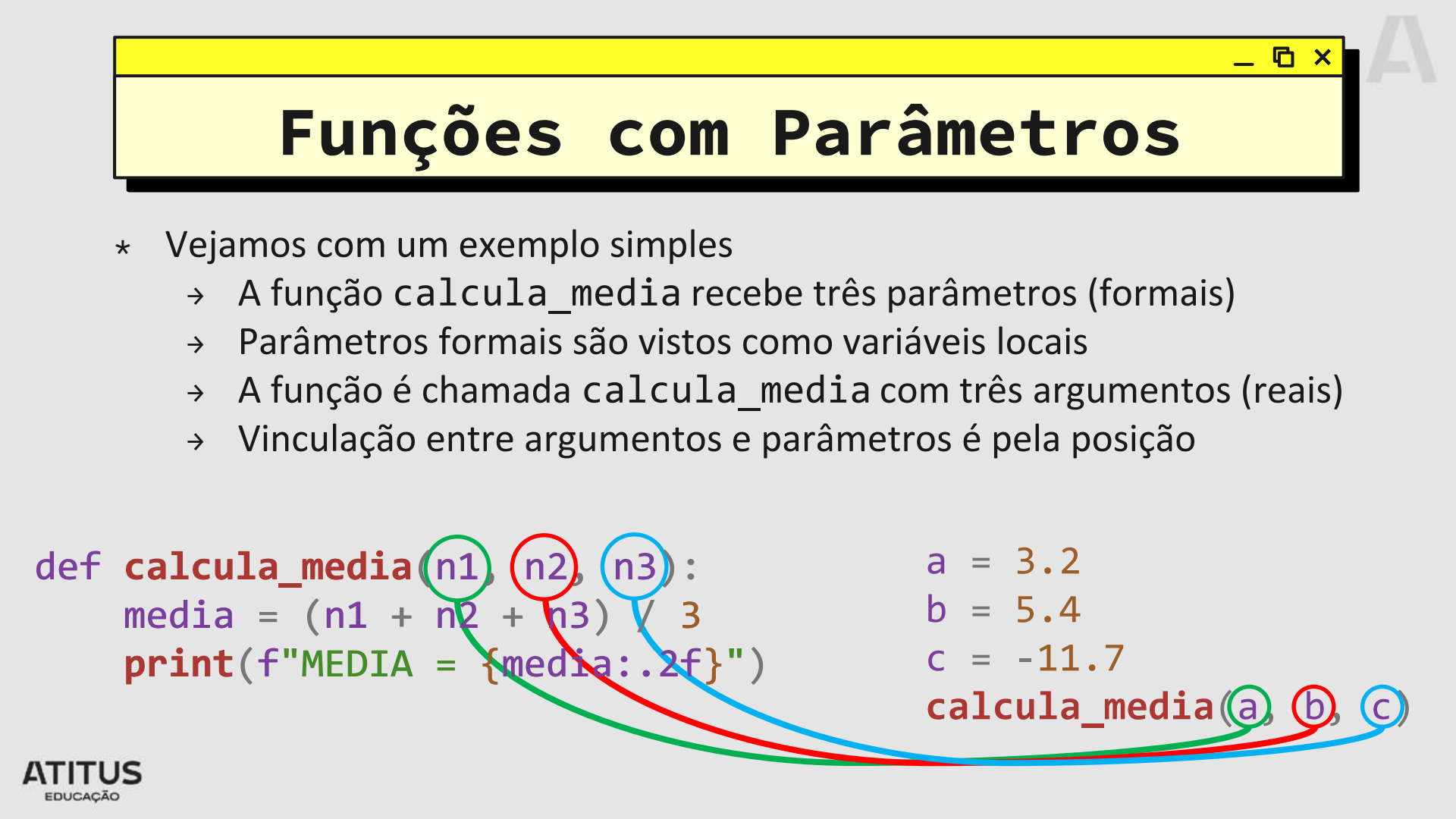
Dados de entrada recebidos pela função. Podemos usar uma lista de argumentos separados por vírgula.

Funções com Parâmetros

* Vejamos com um exemplo simples
  - A função `calcula_media` recebe três parâmetros (formais)
  - Parâmetros formais são vistos como variáveis locais
  - A função é chamada `calcula_media` com três argumentos (reais)
  - Vinculação entre argumentos e parâmetros é pela posição

```
def calcula_media(n1, n2, n3):
    media = (n1 + n2 + n3) / 3
    print(f"MEDIA = {media:.2f}")
```

```
a = 3.2
b = 5.4
c = -11.7
calcula_media(a, b, c)
```



Problema 2: Imprimir n vezes

Faça uma função void que receba 2 parâmetros: um caractere c e um inteiro n. O procedimento deverá imprimir na tela n vezes o caractere c.

Utilize essa função para imprimir o seguinte padrão na tela:

```
a
bb
ccc
dddd
eeeee
ffffff
ggggggg
hhhhhhhh
iiiiiiiii
jjjjjjjjjj
kkkkkkkkkkk
-> lllllllllll
mmmmmmmmmmm
nnnnnnnnnn
ooooooooo
pppppppp
qqqqqqq
rrrrrr
sssss
tttt
uuu
vv
w
```

Problema 2: Imprimir n vezes

```
def imprimir(caracter, vezes):
    print(caracter * vezes)


char = 'a'
repeticoes = 1
incremento = 1
while repeticoes > 0:
    imprimir(char, repeticoes)
    repeticoes += incremento
    char = chr(ord(char) + 1)

    if char == 'l':
        print('-> ', end="")
        incremento = -1
```

Problema 3: Imprimir intervalo de caracteres

Escreva uma função para imprimir os caracteres da tabela ASCII entre 2 inteiros (recebidos como parâmetros).

```
imprime_char(97,140)
```

Problema 3: Imprimir intervalo de caracteres

```
def imprime_char(inicio, fim):
    letras = []
    for val in range(inicio, fim + 1):
        char = chr(val)
        letras.append(char)
    print(", ".join(letras))

ini = int(input("Inicio do intervalo: "))
fim = int(input("Final do intervalo: "))

if ini < fim:
    imprime_char(ini, fim)
else:
    imprime_char(fim, ini)
```

Funções com Retorno

* Tem como objetivo devolver/retornar um valor para o programa principal (ou função) que realizou a chamada

* O valor pode ser:
  - → Valor constante
  - → Valor de uma variável dentro da respectiva função
  - → Resultado de uma expressão lógica ou aritmética

* O valor retornado precisa ser "recebido" no programa principal (ou função) que realizou a chamada através de uma atribuição ou pelo seu uso em uma expressão

Funções com Retorno

* Fazemos uso da palavra reservada `return`, seguida do valor a ser retornado, para retornar um valor de uma função

```
def nome(arg1, arg2, …, argN):
    <comando>
    <comando>
    <comando>
    return valor
```

Comando indicando retorno do valor a seguir

Valor constante, variável ou expressão a ser retornado pela função

Comando return

* Uma função termina sua execução quando seu último comando for executado ou quando ela executar o comando return.

* O comando return provoca a saída imediata da função que o contém e retoma a execução da função chamadora

* Comando return devolve os valores que o seguem para a função chamadora.

Funções com Retorno

```
def nome(arg1, arg2, …, argN):
    ...
    return valor
```

Declaração da função

No caso, o retorno está sendo feito a partir do valor atribuído para a variável **valor** no escopo da função.

```
# Programa principal
...
r = nome(a1, a2, …, aN)
...
```

Chamada da função

Nesse caso, o valor retornado pela função está sendo **atribuído à variável r** no programa principal.

Funções com retorno – utilização

A chamada de uma função com retorno pode ser incluída em uma expressão, atribuição ou função de impressão do programa.

```python
def soma(a, b):
    return a + b
```

```python
v1 = 10
v2 = 15

soma_val = soma(v1, v2)

if (soma(v1, v2) == 25):
    print("Somou 25!")

print(f"A soma é {soma(v1, v2)}")
```

Problema 4: Menu de Opções com retorno

Reescreva a função menu_de_opcoes, para que faça a leitura da opção informada. Se o valor digitado for válido, retorne o valor lido. Caso o valor seja inválido, retorne -1.

```
----------
   MENU
----------

  1 - Soma de dois valores reais
  2 - Divisores do número
  3 - Sequência de números pares
  4 - Verifica se o número é perfeito

Informe a opção desejada:
        ?
```

No programa principal, inclua a chamada desta função e escreva qual a opção foi escolhida (valor válido) ou uma mensagem de erro (valor inválido).

Problema 4: Menu de Opções com retorno

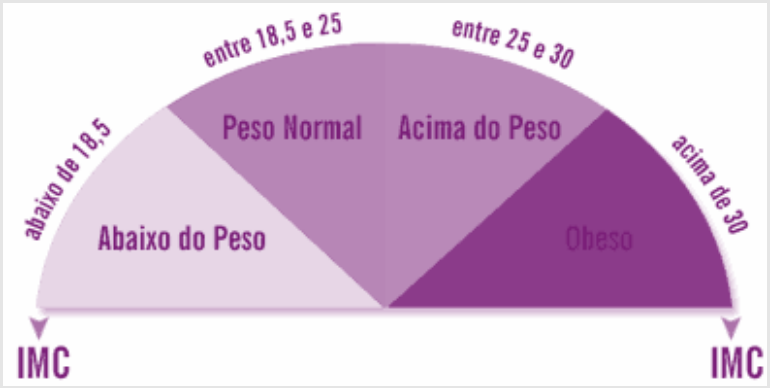
```python
def menu_de_opcoes():
    print("-" * 10)
    print("MENU".center(10))
    print("-" * 10)
    print()
    print("  1 - Soma de dois valores reais")
    print("  2 - Divisores do número")
    print("  3 - Sequência de números pares ")
    print("  4 - Verifica se o número é perfeito")

    print("\nInforme a opção desejada:")
    opcao = int(input("\t? "))
    if opcao >= 1 and opcao <= 4:
        return opcao
    else:
        return -1
```

```python
op = menu_de_opcoes()
if op != -1:
    print(f"Opção escolhida: {op}")
else:
    print("Opção Inválida!")
```

Problema 5: IMC

Faça um programa para calcular o IMC de uma pessoa, mostrando uma mensagem de acordo com a tabela abaixo.
Escreva uma função chamada `imc` que deve receber o peso e altura e que retorna o valor do IMC calculado.
Chame a função a partir do programa principal, após solicitar os dados do usuário e usando o resultado da função mostre a mensagem apropriada.

$$IMC = \frac{peso}{altura^2}$$



```python
def imc(peso, altura):
    i = peso / (altura ** 2)
    return i

# Programa Principal
p = float(input("Digite seu peso em Kg: "))
a = float(input("Digite sua altura em m: "))

i = imc(p, a)

print(f"Seu IMC é {i:.2f} - Você está
if i < 18.5:
    print("abaixo do peso")
elif i <= 25:
    print("com peso normal")
elif i <= 30:
    print("acima do peso")
else:
    print("obeso")
```

A variável **i** tem seu valor atribuído no escopo da função e esse valor é retornado.

Na chamada, o retorno da função `imc` é atribuído para uma variável **i** definida no escopo do programa principal. Apesar do nome, **não é a mesma variável** do escopo da função. Falaremos de escopo de variáveis mais adiante.

```
Digite seu peso em Kg: 76
Digite sua altura em m: 1.8
Seu IMC é 23.46 - Você está com peso normal
```

Funções – Observações Gerais

Um argumento passado para uma função funciona da mesma forma que uma variável local à função

```
def imc(peso, altura):
    i = peso / (altura ** 2)
    return i
```

Argumentos peso e altura agem como variáveis locais à função

Falaremos detalhadamente de escopo de variáveis na próxima aula!

Funções - Observações Gerais

Os nomes dos argumentos utilizados na declaração de uma função são independentes dos nomes das variáveis usadas para chamar a função

```python
def imc(peso, altura):
    i = peso / (altura ** 2)
    return i

# Programa Principal
p = float(input("Digite seu peso em Kg: "))
a = float(input("Digite sua altura em m: "))

i = imc(p, a)

print(f"Seu IMC é {i:.2f} - Você está ", end="")
if i < 18.5:
    print("abaixo do peso")
elif i <= 25:
    print("com peso normal")
elif i <= 30:
    print("acima do peso")
else:
    print("obeso")
```

p, a, peso e altura independentes!

Funções - Observações Gerais

A quantidade de argumentos usados para chamar uma função deve ser igual a declaração

```
def imc(peso, altura):
    i = peso / (altura ** 2)
    return i


...
i = imc(p, a)
...
```

Chamamos esses argumentos de **posicionais**. Veremos depois que também existem argumentos **nomeados**.

Inserir argumentos a mais ou a menos gera erros do tipo TypeError.

Por exemplo:

```
...
i = imc(p, a , x)
...
```

```
TypeError: imc() takes 2
positional arguments but 3
were given
```

Funções – Observações Gerais

* Qualquer tipo de dado pode ser um argumento de função
  → Inclusive listas, dicionários, etc.

* Qualquer expressão lógica ou aritmética válida pode ser argumento para uma função

* Funções podem chamar outras funções
  → Ao final da execução de uma função o fluxo de execução retorna à função que chamou

* Uma função pode chamar a ela mesma (recursividade)

Problema 6: Fibonacci

Construa uma função que imprima todos os números da sequência de fibonacci, retornando o último valor calculado (da posição que foi pedida).

```
Informe a posição: 6
DENTRO FUNÇÃO: fib(0) = 0
DENTRO FUNÇÃO: fib(1) = 1
DENTRO FUNÇÃO: fib(2) = 1
DENTRO FUNÇÃO: fib(3) = 2
DENTRO FUNÇÃO: fib(4) = 3
DENTRO FUNÇÃO: fib(5) = 5
DENTRO FUNÇÃO: fib(6) = 8
PROGRAMA PRINCIPAL: fib(6) = 5
```

Problema 6: Fibonacci

```python
def fibonacci(posicao):
    t_1 = 1
    t_2 = 0
    for i in range(posicao + 1):
        match i:
            case 0:
                fib = 0
            case 1:
                fib = 1
            case _:
                fib = t_2 + t_1
                t_2 = t_1
                t_1 = fib
        print(f"DENTRO FUNÇÃO: fib({i}) = {fib}")
    return fib
```

```python
pos = int(input("Informe a posição: "))
f = fibonacci(pos)
print(f"PROGRAMA PRINCIPAL: fib({pos}) = {f}")
```

Módulos

* Um módulo consiste em um conjunto de funções definidas, que podem ser usadas posteriormente em outros programas

* Módulos podem ser criados por programadores ou podem estar pré-definidos pela linguagem

Módulos

* Para criar um módulo basta codificar um programa que contenha apenas as declarações das funções desejadas e salvá-lo com a extensão .py (como se fosse um código normal)

* Para chamar um módulo basta utilizar o comando `import` seguido do nome do módulo no início do código (certifique-se de que o módulo se encontra na mesma pasta do código principal)

* Para chamar as funções do módulo utilize `nomeMódulo.nomeFunção`

Módulos

* Ao desenvolver módulos, é importante ter em mente duas principais características:

  → Alta coesão: funções de um módulo devem ser fortemente relacionadas entre si
    - Ex: um módulo matemático deve ter apenas funções matemáticas

  → Baixo acoplamento: módulos devem ser compreendidos por si só, sem depender de outros módulos
    - Ex: um módulo matemático não deve depender do módulo math já existente

Problema 6: Combinação de n elementos p a p

Faça um programa que leia dois valores n e p, e calcule a combinação de n elementos, p a p. Assuma que $n \geq p$.

A fórmula para calcular a combinação é:

$$C_p^n = \frac{n!}{p! * (n-p)!}$$

Escreva duas funções, uma para o cálculo do fatorial, e outra para o cálculo da combinação.

Problema 6: Combinação de n elementos p a p

PLANEJAMENTO:

1. Fatorial de um valor
    a. Recebe um valor
    b. Calcula seu fatorial
    c. Retorna fatorial
2. Combinações de n elementos, p a p
    a. Recebe n e p
    b. Calcula n!/(p! * (n-p)!), chamando a função fatorial
    c. Retorna o valor calculado para quem chamou
3. Programa Principal
    a. Lê os valores n e p
    b. Calcula combinação de n elementos, p a p, utilizando a função combinações

```python
def fatorial(numero):
    fat = 0
    if numero > 0:
        fat = 1
        for i in range(numero, 1, -1):
            fat *= i
    return fat

def combinacoes(n, p):
    return fatorial(n) / (fatorial(p) * fatorial(n - p))

n = int(input("Informe o valor de n (número de elementos): "))
p = int(input("Informe o valor de p (qtde elementos por grupo): "))

comb = combinacoes(n, p)
print(f"Combinações de {n} valores {p} a {p} = {comb}")
```

Problema 6: Combinação de n elementos p a p

Modularização:

Podemos separar nossas funções em um módulo diferente da função main do programa.

Para utilizar as funções na função principal, devemos incluir o módulo através da diretiva `import`.

Arquivo: matemática.py

```
def fatorial(numero):
    fat = 0
    if numero > 0:
        fat = 1
        for i in range(numero, 1, -1):
            fat *= i
    return fat

def combinacoes(n, p):
    return fatorial(n) / (fatorial(p) * fatorial(n - p))
```

Programa Principal

```
import matematica

n = int(input("Informe o valor de n (número de elementos): "))
p = int(input("Informe o valor de p (qtde elementos por grupo): "))

comb = matematica.combinacoes(n, p)
print(f"Combinações de {n} valores {p} a {p} = {comb}")
```

Problema 7: Cálculo de e^x

Calcule o valor de e^x usando a série abaixo:

$$e^x = \frac{x^0}{0!} + \frac{x^1}{1!} + \frac{x^2}{2!} + \cdots$$

onde x é um valor real, lido do teclado. Os termos devem ser inseridos enquanto forem maiores do que 0.0001 (em valor absoluto)

Funções envolvidas:

`potencia (float x, int n)` (não pode usar a função pow)
`fatorial (int n)` (já feita)
`EnaX (float x)`