

PYTHON BÁSICO

Prof. Peter Jandl Junior

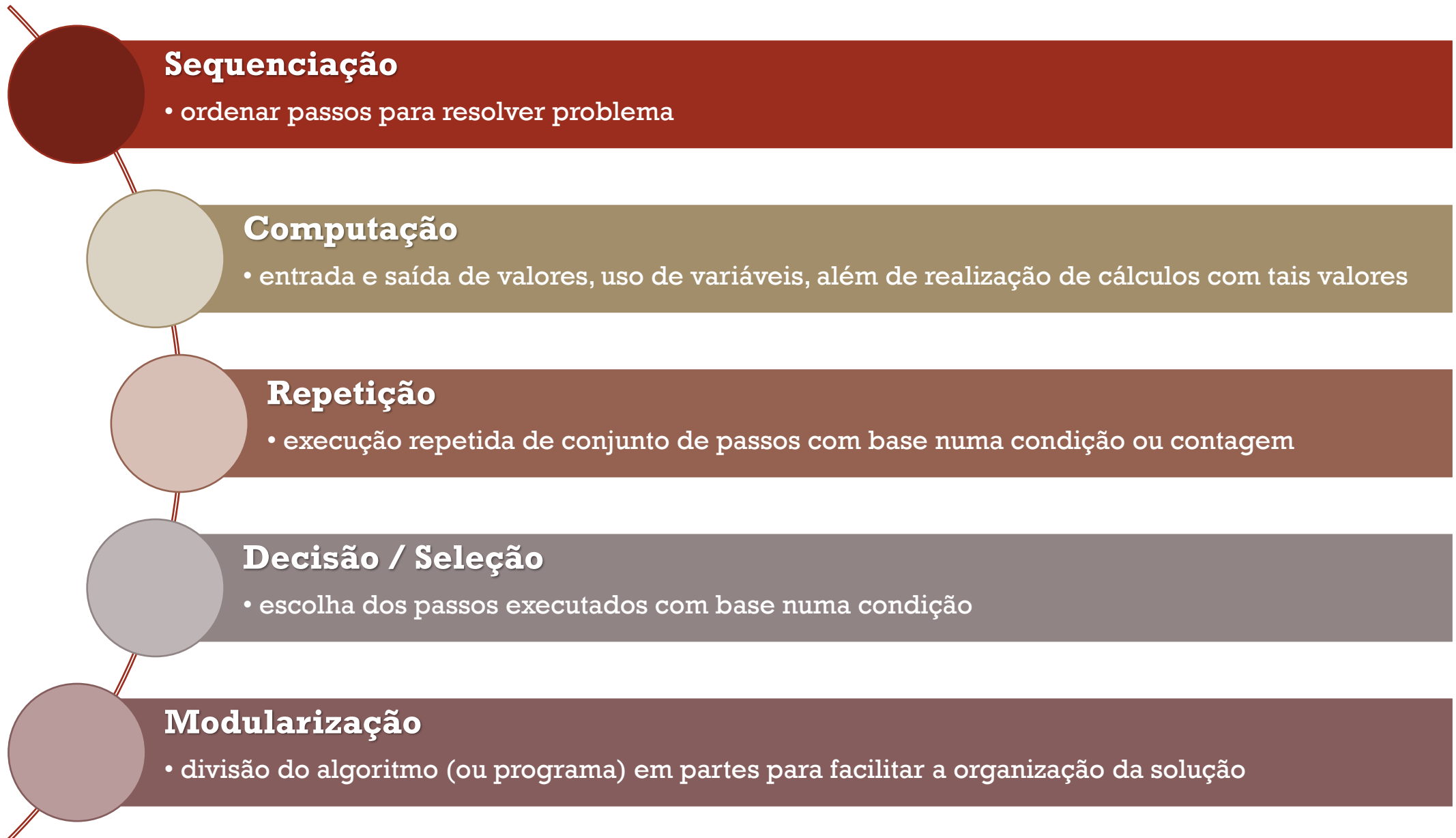
3

DIA 3 [ROTEIRO]



Not so Long ago
In a galaxy that isn't very far away....

HABILIDADES PARA CONSTRUÇÃO DE ALGORITMOS E PROGRAMAS





5

FUNÇÕES

(C) 1999-2020, Jandl.

29/10/2020

FUNÇÕES

- Uma função é um ***trecho de código independente***, um fragmento especial do programa, ou seja, um ***subprograma***.
- Toda função tem:
 - Um **nome** (seu identificador);
 - Um bloco/trecho de **código** (que realiza uma tarefa específica);
 - Um conjunto de **parâmetros** opcionais (que podem customizar a realização da tarefa);
 - E um **resultado**, opcional, (que pode ser um valor de qualquer tipo).

FUNÇÕES

- Construir funções é uma prática tanto típica quanto essencial na programação.



At last we will reveal ourselves to the Jedi.
At last we will have revenge.

FUNÇÕES::DEFINIÇÃO



def nomeDaFuncao (Lista_Param) :

diretivas
return [valor]

Os comandos contidos na definição de uma função são aqueles indentados à direita (padrão é 4 espaços)

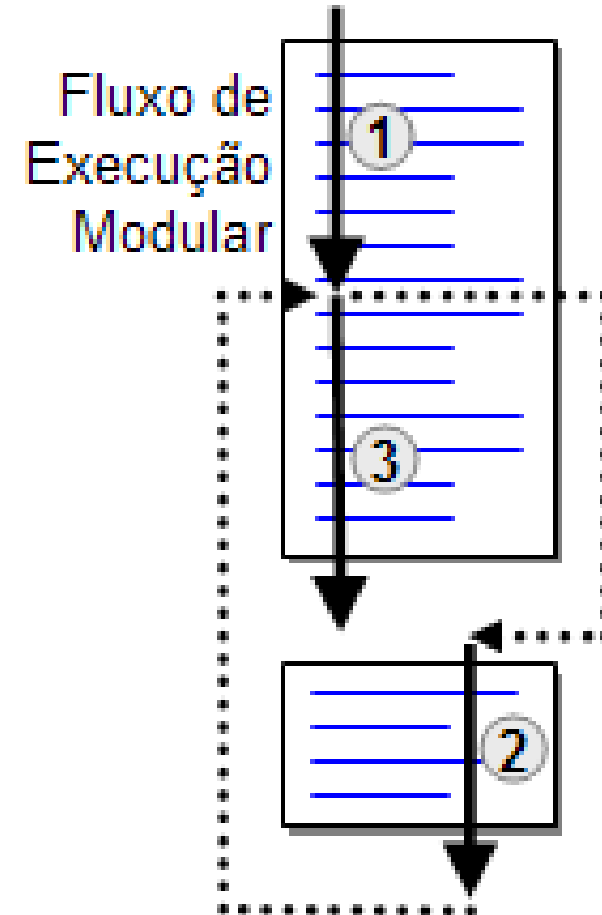
- Onde:
 - **Nome:** identificador da função
 - **Lista_Param:** lista contend zero um ou mais parâmetros da função
 - **Diretivas:** conjunto de diretivas que executam a tarefa da função
 - **return:** indica o término da função e, opcionalmente, seu resultado.

FUNÇÕES E SUA CHAMADA

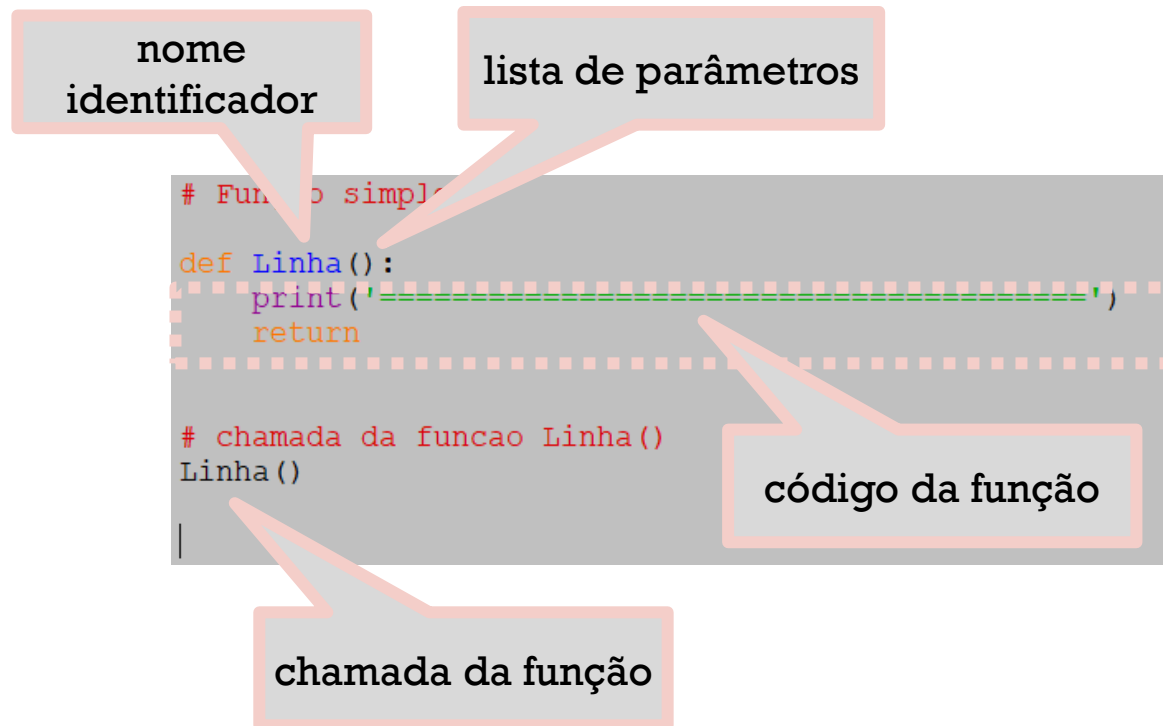
- As funções são executadas por meio de sua **chamada** (*function call*), que pode ocorrer em qualquer ponto do programa, repetidas vezes.
- A chamada de uma função é feita com:
 - Uso de seu **nome** (identificador);
 - Acompanhado dos **argumentos** requeridos.

Ao construir funções, o programador torna seu programa modular!

- (1) Execução sequencial
- (2) Chamada da função (desvio na execução)
- (3) Retorno de valor e continuação da execução.



FUNÇÕES E SUA CHAMADA



- Funções sem parâmetros não tomam argumentos em sua chamada.
- Funções sem retorno de valor são como comandos que produzem um efeito. Não há resultado para atribuir à variáveis ou utilizar em expressões.

FUNÇÕES E SUA CHAMADA

nome
identificador

lista de parâmetros

```
# Função simples
def potenciaDeDois(n):
    resultado = 2**n
    return resultado

# chamada da função potenciaDeDois()
x = potenciaDeDois(20)

print(potenciaDeDois(8))
|
```

código da função

chamada da função

- Funções com parâmetros requerem um argumento (um valor) para cada um de seus parâmetros em sua chamada.
- Funções com retorno de valor são como comandos que devolvem um resultado.
- É comum atribuir tal resultado à variáveis ou utilizá-lo em expressões.

S

R

M

Duas
linhas
em
branco

```
*funcao_simples.py - C:\Users\pjand\Desktop\Oficina Python\código-fonte\Dia_3\funcao_simples.py (3.8.0)*
File Edit Format Run Options Window Help
# Função simples
'''
Nome:          Linha
Propósito:     imprimir uma linha no console
Parâmetros:    não tem
Valor de retorno: não tem
'''
def Linha():
    print('=====')
    return

#
# Programa "principal" segue após as definições das funções
#
print('Exemplo de Programa com Função Simples')
# chamada da funcao Linha()
Linha()
print('Funções podem ser acionadas várias vezes')
for i in range(0, 5):
    print(i, ".", sep='', end='')
    Linha()

print('Final do programa')
Linha()
|
```

Boa prática: documente,
minimamente, suas funções.

Return indica o final da função.

Observe o uso avançado de print!

A função Linha() é usada
várias vezes no programa.



FUNÇÃO SIMPLES

- Funções podem realizar qualquer tipo de tarefa.
- Devem ser definidas **antes** de sua utilização.
- Devem ser separadas por **duas linhas** em branco, umas das outras (padrão Python).
- Devem ser documentadas.
- Podem ser utilizadas inúmeras vezes num programa!

FUNÇÕES, PARÂMETROS E RETORNO DE VALOR

Função produtora

Sem
Parâmetros
Com
Retorno

Com
Parâmetros
Com
Retorno

Função matemática típica

Procedimento

Sem
Parâmetros
Sem
Retorno

Com
Parâmetros
Sem
Retorno

Função consumidora

14



VALOR DE RETORNO

FUNÇÕES

Chamada da função

- O acionamento de uma função é mais conhecido como chamada de função.
- A chamada da função usa seu nome, acompanhado, obrigatoriamente, de parêntesis:
 - `nomedafuncao ()`
- Dentro dos parêntesis devem ser indicados os argumentos requeridos pela função:
 - `nomedafuncao (arg1, arg2 ... argN)`
- Os argumentos passados para uma função permitem que ela tenha sua execução customizada.



Retorno de valor

- Funções **podem** produzir um resultado, o qual pode ser devolvido ao ponto de chamada, tornando conveniente sua utilização.
- A diretiva **return** é utilizada para indicar qual o valor a ser devolvido:
 - `return valor`
- O valor pode ser um literal, variável ou expressão.
- O uso de `return` sempre indica o término da função.

Código
idêntico!

```
entradavalidada.py - C:/Users/pjand/Desktop/Oficina Python/código-fonte/Dia_2/entradavalidada.py (3.8.0)
File Edit Format Run Options Window Help
# Validação de entrada
# Quando é necessário ler um valor DENTRO de uma faixa de valores
MINIMO = 0.0 # valor mínimo da faixa (incluso)
MAXIMO = 10.0 # valor máximo da faixa (incluso)

# Entrada de dados
nota1 = float(input('Digite 1a nota [0.0, 10.0]: '))
# Validação: repete se nota1 FORA da faixa
while nota1 < MINIMO or nota1 > MAXIMO:
    # Informa usuário sobre seu erro
    print('Êita! Vamos tentar novamente!')
    # Repete leitura
    nota1 = float(input('Digite 1a nota [0.0, 10.0]: '))
# Comando executado após laço, então nota1 ESTÁ na faixa

nota2 = float(input('Digite 2a nota [0.0, 10.0]: '))
# Validação: repete se valor FORA da faixa
while nota2 < MINIMO or nota2 > MAXIMO:
    # Informa usuário sobre seu erro
    print('Êita! Vamos tentar novamente!')
    # Repete leitura
    nota2 = float(input('Digite 2a nota [0.0, 10.0]: '))
# Comando executado após laço, então nota2 ESTÁ na faixa

# Saída de dados
media = (nota1 + nota2) / 2
print('Nota1: {:.1f} | Nota2: {:.1f} | Media {:.2f}'
      .format(nota1, nota2, media))
```

Repetição
não é
conveniente!

APLICAÇÃO DE FUNÇÕES

- Um dos melhores usos das funções é evitar a repetição de código.
- Neste exemplo, a entrada e dados e sua validação utiliza praticamente o mesmo código.
- A identificação do código comum (fatoração) permite criar uma função adequada.



Duas
linhas
em
branco

```
*entradavalidada2.py - C:\Users\pjand\Desktop\Oficina Python\código-fonte\Dia_3\entradavalidada2.py (3.8.0)*
File Edit Format Run Options Window Help

# Validação de entrada com função

'''
Nome:          LeituraValidada
Propósito:     Efetua leitura validada na faixa [MINIMO,MAXIMO]
Parâmetros:    não tem
Valor de retorno: real
'''

MINIMO = 0.0    # valor mínimo da faixa (incluso)
MAXIMO = 10.0   # valor máximo da faixa (incluso)

def LeituraValidada():
    nota = float(input('Digite a nota [{:.1f},{:.1f}]: '
                       .format(MINIMO,MAXIMO)))
    while nota < MINIMO or nota > MAXIMO:
        # Informa usuário sobre seu erro
        print('Valor inválido! Repita por favor.')
        # Repete leitura
        nota = float(input('Digite a nota [{:.1f},{:.1f}]: '
                           .format(MINIMO,MAXIMO)))
    # Comando executado após laço, então notal ESTÁ na faixa
    return nota    # retorno de valor (a nota)

#
# Programa principal
#
# Entrada de dados
notal = LeituraValidada()
nota2 = LeituraValidada()

# Saída de dados
media = (notal + nota2) / 2
print('Notal: {:.1f} | Nota2: {:.1f} | Media {:.2f}'
      .format(notal, nota2, media))
|
```

Boa prática!

Função elimina
repetição do código

Programa principal se torna
mais simples, pois a função é
como um novo comando!

APLICAÇÃO DE FUNÇÕES

- Um dos melhores usos das funções é evitar a repetição de código.
- A função **LeituraValidada()** evita a repetição de código e permite ler valores validados.
- Seu nome torna-se uma abstração da tarefa que realiza, como **print()** ou **input()** ou outras funções.
- Usar funções reduz e simplifica o código!



VANTAGENS DO USO DE FUNÇÕES

- **Abstração/Entendimento:** é mais fácil entender vários trechos independentes de código do que um programa grande.
- **Organização:** o código usado repetidas vezes fica "separado" de sua aplicação.
- **Reuso:** funções podem ser incluídas numa biblioteca e reutilizada em programas diferentes.
- **Manutenção:** como a construção de funções evita a repetição de código, a manutenção fica concentrada na função em si.
- **Tamanho:** programas menores (memória e disco), pois o código deixa de ser repetido.



S

C

D

M

Duas
linhas
em
branco

```

dados.py - C:/Users/pjand/Desktop/Oficina Python/código-fonte/Dia_3/dados.py (3.8.0)
File Edit Format Run Options Window Help
# Uso Função de geração de valor para dado

import random

'''
Nome:          dado
Propósito:     Gera um valor aleatório entre [1,6]
Parâmetros:    não tem
Valor de retorno: int
'''

def dado():
    numero = random.randrange(1,7)
    return numero      # retorno de valor (numero "obtido" no dado)

#
# Programa principal
#
print('Lançamento de Dados')
print('-----')
dado1 = dado()
dado2 = dado()
print('dado1 :', dado1, '\ndado2 :', dado2)
print('-----')
if dado1+dado2 == 7:
    print('Sete! Você ganhou!')
elif dado1+dado2 == 12:
    print('Doze! Você está com sorte!')
else:
    print('Boa tentativa.')
|

```

Importação

Função elimina
repetição do código

Programa principal se torna
mais simples, pois a função é
como um novo comando!



APLICAÇÃO DE FUNÇÕES

- Outro exemplo de função, que presta um serviço, isto é, realiza uma tarefa específica.
- A função **dado()** evita a repetição de código e se torna uma abstração do lançamento de um dado.
- Usar funções sempre reduz e simplifica o código!



THE ZEN OF PYTHON

- Bonito é melhor que feio.
- Explícito é melhor que implícito.
- Simples é melhor que complexo.
- Complexo é melhor que complicado.
- Esparso é melhor que denso.
- Legibilidade conta.
- Casos especiais não são especiais o suficiente para quebrar as regras.

Beautiful is better than ugly.
Explicit is better than implicit. Simple is better than complex. Complex is better than complicated. Flat is better than nested. Sparse is better than dense. Readability counts. Special cases aren't special enough to break the rules.

Although practicality beats purity. Errors should never pass silently. Unless explicitly silenced. In the face of ambiguity, refuse the temptation to guess. There should be one — and preferably only one — obvious way to do it. Although that way may not be obvious at first unless you're Dutch. Now is better than never. Although never is often better than right now. If the implementation is hard to explain, it's a bad idea. If the implementation is easy to explain, it may be a good idea. Namespaces are one honking great idea — let's do more of those!

Beautiful is better than ugly.
Explicit is better than implicit. Simple is better than complex. Complex is better than complicated. Flat is better than nested. Sparse is better than dense. Readability counts. Special cases aren't special enough to break the rules. Although practicality beats purity. Errors should never pass silently. Unless explicitly silenced. In the face of ambiguity, refuse the temptation to guess. There should be one — and preferably only one — obvious way to do it. Although that way may not be obvious at first unless you're Dutch. Now is better than never. Although never is often better than right now. If the implementation is hard to explain, it's a bad idea. If the implementation is easy to explain, it may be a good idea. Namespaces are one honking great idea — let's do more of those!



21

PASSAGEM DE PARÂMETROS

(C) 1999-2020, Jandl.

29/10/2020

PARÂMETROS

- Uma função pode ter sua execução modificada com o uso de parâmetros, ou seja, os *argumentos recebidos como parâmetros podem modificar o comportamento da função*.
- Funções podem ser definidas com ***nenhum, um ou muitos parâmetros***.
- A relação de parâmetros necessários por uma função é conhecida como ***lista de parâmetros***.

Os **parâmetros** pode ser de qualquer tipo!

- Cada parâmetro indicado na lista de parâmetros é como uma variável local (interior) à função.

lista de parâmetros é:
a e b.

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

função sempre executa soma de **a + b**.

resultado retornado depende dos valores de a e b.

- Os valores dos parâmetros, isto é, seus ***argumentos***, são fornecidos no instante da chamada da função:

```
y = 13.5  
z = soma( 6.5, y ) # chamada da função
```

Os **argumentos** devem ser compatíveis com o esperado pela função!

S

C

R

D

M

Duas
linhas
em
branco

Funções com um parâmetro e
retorno de valor.

Faltou documentar!

```
funcoes_parametrizadas.py - C:/Users/pjand/Desktop/Oficina P.../Dia_3/funcoes_parametrizadas.py (3.8.0)
File Edit Format Run Options Window Help
# Funções parametrizadas

def ePar(valor):
    resultado = valor % 2 == 0 # testa se resto da divisão inteira por 2 é zero
    return resultado          # retorna resultado lógico (bool) do teste

def ePrimo(valor):
    if valor < 2: return False # se valor menor que 2 não pode ser primo
    for i in range(2, valor): # verifica divisão entre 2 e valor
        # testa se resto da divisão inteira de valor por i é zero (divisível)
        if valor % i == 0: return False
    return True                # valor é primo

#
# Programa principal
#
print('Testes')
print('-----')
valor = int(input('Digite um inteiro positivo (>0): '))
print('-----')
# chamada de função, COM armazenamento do resultado
par = ePar(valor)
print(valor, 'é par: ', par)

# chamada de função, SEM armazenamento do resultado
print(valor, 'é primo: ', ePrimo(valor))
```

Uso direto da
função.

Uso da função
em uma função.



FUNÇÕES PARAMETRIZADAS

- Funções podem:
 - Declarar variáveis;
 - Computar expressões;
 - Realizar repetição;
 - Realizar decisão; e
 - Usar outras funções!

- *Usar funções é uma boa prática de programação independente da linguagem utilizada!*

```
media.py - C:/Users/pjand/Desktop/Oficina Python/código-fonte/Dia_3/media.py (3.8.0)
File Edit Format Run Options Window Help
# Funções parametrizadas
import math

def mediaAritmetica(a, b):
    resultado = (a + b) / 2
    return resultado        # retorna a média aritmética de a e b

def mediaGeometrica(a, b):
    resultado = math.sqrt(a*b)
    return resultado        # retorna a média geométrica de a e b

#
# Programa principal
#
print('Médias')
print('-----')
valor1 = float(input('Valor real 1? '))
valor2 = float(input('Valor real 2? '))
print('-----')
media = mediaAritmetica(valor1, valor2)
print('Media Aritmetica({:.2f},{:.2f}) = {:.2f}'
      .format(valor1, valor2, media))

print('Media Geometrica({:.2f},{:.2f}) = {:.2f}'
      .format(valor1, valor2, mediaGeometrica(valor1, valor2)))
```

Importação do módulo math.

Faltou documentar!

Funções com dois parâmetros e retorno de valor.

Observe o uso da função importada do módulo math!

Uso direto da função.

Uso da função em uma função dentro de outra função!



FUNÇÕES PARAMETRIZADAS

- Funções parametrizadas organizam o programa e permitem construir elementos que poderão ser usados em outros programas.
- *Usar funções é uma boa prática de programação independente da linguagem utilizada!*

Duas
linhas
em
branco



PARÂMETROS DEFAULT

- É possível atribuir valores padrão (default) para parâmetros.
- ***Parâmetros default*** não precisam ***argumentos*** (ser indicados) na chamada da função.
- Os *argumentos* correspondentes aos parâmetros default *podem ser supridos opcionalmente* na chamada da função, quando se deseja usar valores diferentes do padrão.
- Flexibiliza muito as possibilidades de uso de uma função.

```
" Verifica situação de aprovação numa disc.  
Retorna: 0 (aprovado), 1 (reprov. nota),  
2 (reprov. falta) ou 3 (reprov. nota e falta)  
"
```

```
def situacao(nota, freq, CH, MAp=6.0, FAp=.75):  
  
    situacao = 0  
  
    if nota < MAp:  
        situacao += 1  
  
    if (freq/CH) < FAp:  
        situacao += 2  
  
    return situacao
```

S

C

D

M

Duas
linhas
em
branco

```

situacao.py - C:\Users\pjand\Desktop\Oficina Python\código-fonte\Dia_3\situacao.py (3.8.0)
File Edit Format Run Options Window Help

'''
Nome:          situacao
Propósito:     Verifica situação de aprovação numa disciplina
Parâmetros:    nota(float), frequencia(float), carga horária(int)
                média aprovação(float=6.0), frequência aprovação(float=0.75)
Valor de retorno: int 0 (aprovado), 1 (reprov. nota),
                2 (reprov. falta) ou 3 (reprov. nota e falta)
'''

def situacao(nota, freq, CH, MAP=6.0, FAp=.75):
    situacao = 0
    if nota < MAP:
        situacao += 1
    if (freq/CH) < FAp:
        situacao += 2
    return situacao

#
# Programa principal
#
print('Situação de Aprovação')
print('-----')
nota = float(input('Digite sua nota (real): '))
freq = int(input('Digite suas presenças (inteiro): '))
ch = int(input('Digite carga horária (inteiro): '))
print('-----')
print('Padrão: média 6.0 e 75% freq. mínima')
print('Situação: ', situacao(nota, freq, ch))
print('-----')
print('Especial: média 5.0 e 80% freq. mínima')
print('Situação: ', situacao(nota, freq, ch, 5.0, 0.8))
print('-----')

```

Função documentada!

Função com dois
parâmetros default.

Observe o uso da função
importada do módulo math!

Uso simples da função.

Uso da função com novos valores
para os parâmetros default.



FUNÇÕES COM PARÂMETROS DEFAULT

- Funções com parâmetros default pode ser usadas:
 - Diretamente (só fornecendo os argumentos obrigatórios);
 - Indicando um, dois ou mais dos argumentos opcionais (sempre na ordem!).
- *O print é uma função que tem parâmetros default, para seu separador e terminador de linha!*

PARÂMETROS VARIÁVEIS

- É possível que uma função receba um número variável de parâmetros.
- O parâmetro indicado com um prefixo * indica esta opção.
- Podem ser passados 0, 1, 2 ou tantos parâmetros quantos desejados.

```
def media(*valor):
```

Parâmetro variável

```
    soma = 0
```

```
    quant = 0
```

```
    for v in valor:
```

Argumentos recebidos
como uma lista

```
        soma += v
```

```
        quant += 1
```

```
    if quant == 0: return
```

```
    return soma/quant
```

**O parâmetro variável
é sempre o último da
lista de parâmetros!**



S

C

R

D

M

Duas
linhas
em
branco

```
media2.py - C:/Users/pjand/Desktop/Oficina Python/código-fonte/Dia_3/media2.py (3.8.0)
File Edit Format Run Options Window Help

# Funções com parâmetros variáveis

def mediaAritmetica(*a):
    quant = 0
    soma = 0
    for v in a:
        soma += v
        quant += 1
    if quant == 0: return
    return soma / quant # retorna média aritmética dos argumentos

def mediaGeometrica(*a):
    quant = 0
    produto = 1
    for v in a:
        produto *= v
        quant += 1
    if quant == 0: return
    return produto**(1/quant) # retorna média geométrica dos argumentos

#
# Programa principal
#
print('Médias 2: parâmetros variáveis')
print('-----')
print('Media Aritmetica =', mediaAritmetica())
print('Media Aritmetica =', mediaAritmetica(1))
print('Media Aritmetica =', mediaAritmetica(1, 2.0))
print('Media Aritmetica =', mediaAritmetica(1, 2.3, 4.6, 7.8))
print('-----')
print('Media Geométrica =', mediaGeometrica())
print('Media Geométrica =', mediaGeometrica(1))
print('Media Geométrica =', mediaGeometrica(1, 2.0))
print('Media Geométrica =', mediaGeometrica(1, 2.3, 4.6, 7.8))
print('-----')
```

Parâmetros variáveis

Somatório e contagem dos argumentos

Parâmetros variáveis

Produtório e contagem dos argumentos

Uso da função com 0, 1 2 e mais argumentos

Observe o resultado None quando a função não toma argumentos



FUNÇÕES COM PARÂMETROS DEFAULT

- Funções com parâmetros default pode ser usadas:
 - Diretamente (só fornecendo os argumentos obrigatórios);
 - Indicando um, dois ou mais dos argumentos opcionais (sempre na ordem!).
- O print é uma função que tem parâmetros default, para seu separador e terminador de linha!*



29

MODULARIZAÇÃO

(C) 1999-2020, Jandl.

29/10/2020



(C) 1999-2020, Jandl.

MÓDULO

- Conjunto de definições que podem ser empregadas em um ou mais programas.
- Podem incluir funções e estruturas de dados.
- Um módulo pode ser:
 - **Horizontal:** conter elementos de mesma natureza (funções matemáticas, como Math; ou conversores de unidades).
 - **Vertical:** conter elementos de vários tipos, mas que podem ser combinadas para resolver problemas de um determinado domínio (manipulação de imagens, OCR etc).
- Módulos permitem dividir um problema grande ou complexo em partes menores, que podem ser tratadas (resolvidas) separadamente.

S

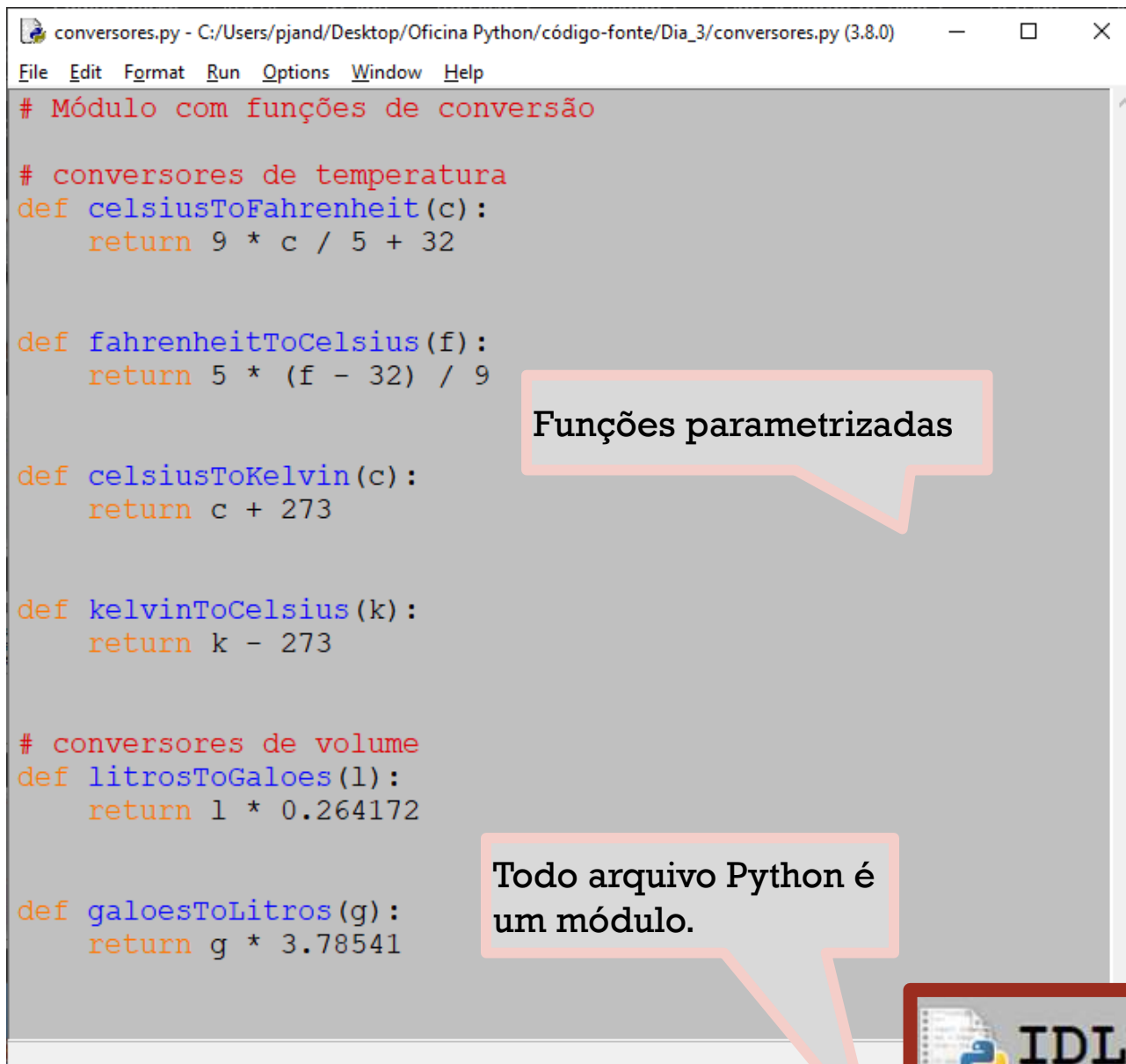
C

R

D

M

Duas
linhas
em
branco



```
conversores.py - C:/Users/pjand/Desktop/Oficina Python/código-fonte/Dia_3/conversores.py (3.8.0)
File Edit Format Run Options Window Help
# Módulo com funções de conversão

# conversores de temperatura
def celsiusToFahrenheit(c):
    return 9 * c / 5 + 32

def fahrenheitToCelsius(f):
    return 5 * (f - 32) / 9

def celsiusToKelvin(c):
    return c + 273

def kelvinToCelsius(k):
    return k - 273

# conversores de volume
def litrosToGaloos(l):
    return l * 0.264172

def galoosToLitros(g):
    return g * 3.78541
```

Funções parametrizadas

Todo arquivo Python é
um módulo.



MÓDULO SIMPLES

- Um módulo simples pode ser constituído de uma ou mais funções, parametrizadas ou não.
- Quanto mais independente forem as funções presentes em um módulo, maiores são as chances de sua reutilização.
- **Reutilização (reuso)** é um dos pilares importantes da engenharia de software, pois representa preservação de investimento!

USO DE MÓDULOS

- Requer sua importação:

import nomeDoModulo

- O uso das funções presentes no módulo requer sua **qualificação**, isto é, usar o *nome do módulo como seu prefixo*.
- O módulo deve estar presente:
 - No mesmo diretório do arquivo Python que o utiliza;
 - Num dos diretórios configurados para conter módulos (*module path*).



```
*teste_modulo.py - C:/Users/pjand/Desktop/Oficina Python/código-fonte/Dia_3/teste_modulo.py (3.8.0)*
File Edit Format Run Options Window Help

# Exemplo de uso de módulo

import conversores # importação é requerida para dar
                  # acesso aos elementos do módulo

#
# Programa Simples
#
print("Conversão de Temperatura")
print("-----")
print("1. Celsius para Farenheit")
print("2. Celsius para Kelvin")
print("3. Farenheit para Celsius")
print("4. Kelvin para Celsius")
print("-----")
opcao = int(input('Opção [1..4]? '))
if (opcao == 1):
    c = float(input('Temperatura C? '))
    f = conversores.celsiusToFahrenheit(c)
```

celsiusToFahrenheit
celsiusToKelvin
fahrenheitToCelsius
galoesToLitros
kelvinToCelsius
litrosToGaloes

No IDLE, basta qualificar para receber a lista de funções disponíveis!

USO DE MÓDULOS

- Requer sua importação:

import nomeDoModulo

- O uso das funções presentes no módulo requer sua **qualificação**, isto é, usar o *nome do módulo como seu prefixo*.
- O módulo deve estar presente:
 - No mesmo diretório do arquivo Python que o utiliza;
 - Num dos diretórios configurados para conter módulos (*module path*).

teste_modulo.py - C:/Users/pjand/Desktop/Oficina Python/código-fonte/Dia_3/teste_modulo.py (3.8.0)

File Edit Format Run Options Window Help

Exemplo de uso de módulo

import conversores # importação é requerida para dar
acesso aos elementos do módulo

Programa Simples
#

print("Conversão de Temperatura")
print("-----")
print("1. Celsius para Farenheit")
print("2. Celsius para Kelvin")
print("3. Farenheit para Celsius")
print("4. Fahrenheit para Kelvin")
print("5. Kelvin para Celsius")
print("6. Kelvin para Fahrenheit")
print("-----")
opcao = int(input('Opção [1..6]? '))
if opcao == 1 or opcao == 2:
 c = float(input('Temperatura C? '))
 if opcao == 1:
 # uso simples da função do módulo
 f = conversores.celsiusToFahrenheit(c)
 print('{:.2f} C --> {:.2f} F'.format(c, f))
 else:
 k = conversores.celsiusToKelvin(c)
 print('{:.2f} C --> {:.2f} K'.format(c, k))
elif opcao == 3 or opcao == 4:
 f = float(input('Temperatura F? '))
 if opcao == 3:
 c = conversores.fahrenheitToCelsius(f)
 print('{:.2f} F --> {:.2f} C'.format(f, c))
 else:
 # uso combinado das funções do módulo
 k = conversores.celsiusToKelvin(conversores.fahrenheitToCelsius(f))
 print('{:.2f} F --> {:.2f} K'.format(f, k))
elif opcao == 5 or opcao == 6:
 k = float(input('Temperatura K? '))
 if opcao == 5:
 c = conversores.kelvinToCelsius(k)
 print('{:.2f} K --> {:.2f} C'.format(k, c))
 else:
 f = conversores.celsiusToFahrenheit(conversores.kelvinToCelsius(k))
 print('{:.2f} K --> {:.2f} F'.format(k, f))
else:
 print('Opcao inválida!')

Importação

Esse programa se preocupa apenas com a entrada e saída dos dados.

A lógica do negócio (conversão de temperaturas) está separada no módulo!

29/10/2020

Ln: 46 Col: 0

MAIS SOBRE USO DE MÓDULOS

- Módulos podem ser renomeados na importação:

import nomeDoModulo as alias

- O alias é um apelido, em geral, uma simplificação para facilitar a **qualificação**.

```
teste2_modulo.py - C:/Users/pjand/Desktop/Oficina Python/código-fonte/Dia_3/teste2_modulo.py (3.8.0)
File Edit Format Run Options Window Help
# Exemplo de uso de módulo

import conversores as tc # importação e renomeação do módulo
#
# Programa Simples
#
print("Conversão de Temperatura")
print("-----")
print("1. Celsius para Farenheit")
print("2. Celsius para Kelvin")
print("3. Farenheit para Celsius")
print("4. Fahrenheit para Kelvin")
print("5. Kelvin para Celsius")
print("6. Kelvin para Fahrenheit")
print("-----")
opcao = int(input('Opção [1..6]? '))
if opcao == 1 or opcao == 2:
    c = float(input('Temperatura C? '))
    if opcao == 1:
        # uso simples da função do módulo
        f = tc.celsiusToFahrenheit(c)
        print('{:.2f} C --> {:.2f} F'.format(c, f))
    else:
        k = tc.celsiusToKelvin(c)
        print('{:.2f} C --> {:.2f} K'.format(c, k))
elif opcao == 3 or opcao == 4:
    f = float(input('Temperatura F? '))
    if opcao == 3:
        c = tc.fahrenheitToCelsius(f)
        print('{:.2f} F --> {:.2f} C'.format(f, c))
    else:
        # uso combinado das funções do módulo
        k = tc.celsiusToKelvin(tc.fahrenheitToCelsius(f))
        print('{:.2f} F --> {:.2f} K'.format(f, k))
elif opcao == 5 or opcao == 6:
    k = float(input('Temperatura K? '))
    if opcao == 5:
        c = tc.kelvinToCelsius(k)
        print('{:.2f} K --> {:.2f} C'.format(k, c))
    else:
        f = tc.celsiusToFahrenheit(tc.kelvinToCelsius(k))
        print('{:.2f} K --> {:.2f} F'.format(k, f))
else:
    print('Opcao inválida!')
```

Importação e renomeação

O programa
continua o
mesmo!

Acesso mais simples às
funções do módulo.

MAIS SOBRE USO DE MÓDULOS

- Importação seletiva, de um ou elementos do módulo:

from nomeModulo import el, ... en

- Uso não requer qualificação.
Exemplo:

```
from math import pi, pow, sqrt
```

```
print('Pi = ', pi)
```

```
print('2.5**4 = ', pow(2.5, 4))
```

```
print('Raiz de 2 = ', sqrt(2))
```



- Importação global também é possível:

from nomeModulo import *

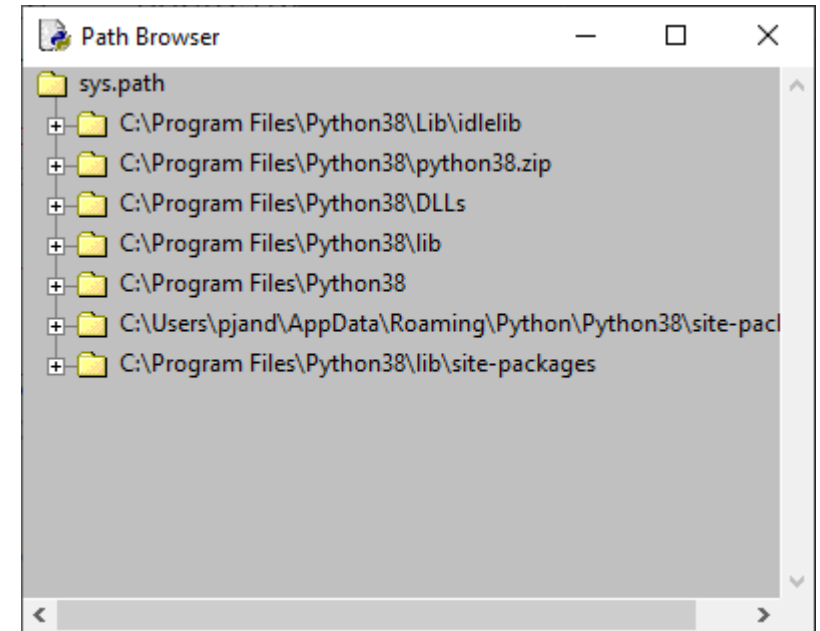
Importação global
não é uma boa
prática!

LOCALIZAÇÃO DO MÓDULOS

- Python busca os módulos:
 - No diretório corrente da execução;
 - Nos diretórios listados na variável de ambiente PYTHONPATH;
 - Na lista de diretórios da instalação local do Python.

- No IDLE:
 - File | Path Browser
 - Ou no modo interativo:

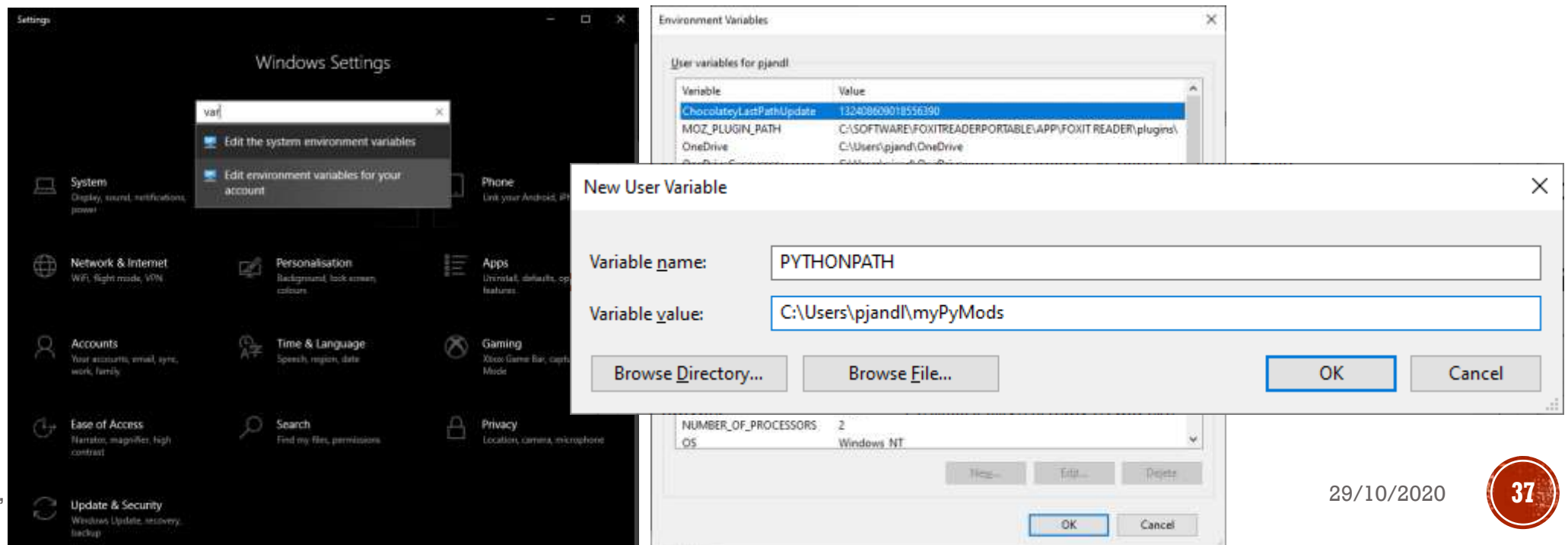
```
import sys  
sys.path
```



```
>>> import sys  
>>> sys.path  
['', 'C:\\Program Files\\Python38\\Lib\\idlelib', 'C:\\Program  
Files\\Python38\\python38.zip', 'C:\\Program Files\\Python38\\  
DLLs', 'C:\\Program Files\\Python38\\lib', 'C:\\Program Files\\  
Python38', 'C:\\Users\\pjand\\AppData\\Roaming\\Python\\Pytho  
n38\\site-packages', 'C:\\Program Files\\Python38\\lib\\site-p  
ackages']  
>>>
```


CONFIGURANDO SEU PYTHONPATH

1. Selecione ou crie um diretório para adicionar módulos Python.
Por exemplo: C:\Users\pjand\myPyMods
2. Crie uma variável de ambiente PYTHONPATH com a indicação do diretório escolhido. No W10: abra o Painel de Controle.





O QUE JÁ SABEMOS FAZER?

- Computar: entrada, saída e cálculos;
- Repetir: condicional e automático;
- Decidir;
- Sequenciar, combinando tudo isso!



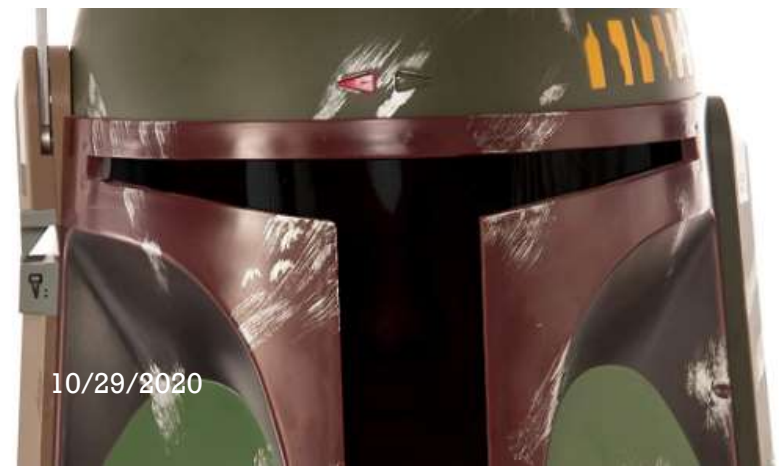
COMPUTAÇÃO

- Entrada de dados com **input()** e as funções de conversão **int()** e **float()**.
- Saída de dados **print()**.
- Definição de **variáveis** e realização de cálculos combinando **valores literais**, **variáveis** e **operadores**.



REPETIÇÃO

- Repetição condicional com **while**.
- Repetição automática com **for**.
- Uso da repetição para **contar** e **acumular**.



DECISÃO

- Seleção de comandos com uso de **if/else**.
- Encadeamento de seleção com **elsif**.
- Permite validar valores e também flexibilizar o uso dos programas.



SEQUENCIACÃO

- Combinação de **computação** (entrada, saída e cálculos), **repetição** e **decisão** para resolver muitos tipos de problemas.



MODULARIZAÇÃO

- Divisão do problema em segmentos específicos e reusáveis que combinam **sequenciação**, **computação**, **repetição** e **decisão** para auxiliar na solução de problemas diferentes.



MÃOS NA MASSA

Que força esteja
com você!

- Resolver a Lista **TRÊS**.
- Como pensar como um Cientista da Computação.
Projeto Panda | IME | USP.
<https://panda.ime.usp.br/pensepy/static/pensepy/index.html>
- Python e Orientação a Objetos
Curso Py-14 | Caelum.
<https://www.caelum.com.br/apostila/apostila-python-orientacao-a-objetos.pdf>
- Python 3.8. Documentação Oficial.
<https://docs.python.org/3/index.html>