NIVELAMENTO

ALGORITMO E PROGRAMAÇÃO

LÓGICA COMPUTACIONAL

MODÚLO 1
CONCEITOS BÁSICOS E LÓGICA COMPUTACIONAL

UniSENAI

MODÚLO 1

SUMÁRIO

CONCEITOS BÁSICOS 4
VARIÁVIES 5
TIPOS DE DADOS 6
OPERADORES E EXPRESSÕES 7
Conceito Teórico 7
Operadores Aritméticos 8
Operadores Realacionais 8
Operadores Lógicos 9
ENTRADA E SAÍDA DE DADOS 10
Entrada de Dados 10
Saída de Dados – print( ) 11
01. Uso Básico de print( ) 11
02. Imprimindo Múltiplos Valores 11
03. Concatenação de Strings com + 11
04. Uso de f-strings (Formatação moderna) 11
05. Quebra de Linha e Caracteres Especiais 11
06. Mudança de Final (end) 11
07. Separador de Valores (sep) 12
08. Print com Expressões 12
09. Uso com Strings de Múltiplas Linhas 12
Resumo das Boas Práticas 13
A LÓGICA DO COTIDIANO 14
Objetivo 14
Lógica na Vida Real 14
Exemplo Prático: Decisão dobre Levar o Guarda-Chuva 15
LÓGICA COMPUTACIONAL 16
Conceito Teórico 17
Proposição 17
Encadeamento com elif 18
Pensamento Lógico 18
LÓGICA PROPOSICIONAL E DESVIOS CONDICIONAIS 19
LÓGICA DE PROPOSIÇÕES E TABELA-VERDADE 20
Proposição Simples 20
Conectivos Lógicos 21
Tabela-Verdade 21
Operador AND (and) 21
Operador OR (or) 22
Operador NOT (not) 22
Proposição Composta 23
Desvios Condicionais 23
Simples (if) 24
Com else 24
Múltiplos caminhos (if/elif/else) 24
BANCO DE QUESTÕES 26

MÓDULO I:

CONCEITOS BÁSICOS E LÓGICA COMPUTACIONAL

Neste primeiro momento, vamos estudar os conceitos básicos, mas fundamentais da programação de algoritmos como os tipos de dados, variáveis, operadores, expressões, entrada e saída de dados e o fantástico mundo da lógica computacional e seus principais aspectos!

Estou muito animado para iniciarmos nossos estudos. E para aproveitar ao máximo o encontro síncrono da semana, fique atento e siga as etapas abaixo:

☑Leia a apostila para compreender os conceitos fundamentais e a lógica computacional.
☑Assista às videoaulas que selecionamos para reforçar o aprendizado.
☑Pratique com o banco de algoritmos, a lista de exercícios e os questionários de estudo e aplique os conceitos na prática para a resolução de problemas.

Dica: Anote suas dúvidas para discutirmos no encontro síncrono!

- Revisitando conceitos básico: tipos de dados, variáveis, operadores, expressões, entrada e saída de dados
- Lógica, Raciocínio lógico e Lógica de proposições e Estruturas condicionais
- Exercícios práticos

NIVELAMENTO

CONCEITOS BÁSICOS

Comentário

Comentário é uma parte do código que é inserido apenas para explicar algum detalhe que o programador considerar importante. Ele não é executado pelo computador e serve para explicar o que o código faz, facilitar a leitura e a manutenção do programa.
Em Python:

- Comentários começam com o símbolo #.
- Tudo o que vem depois do # na mesma linha é ignorado pelo interpretador.

Atenção: Neste material vamos utilizar esse recurso para complementar alguma explicação sobre o código e o resultado da sua execução.

Por que usar comentários?

- Para explicar a lógica do algoritmo, principalmente em trechos complexos.
- Para lembrar o que você fez, caso precise revisar o código depois.
- Para ajudar outras pessoas a entenderem o que o programa está fazendo.

Dica de boas práticas:

- Comente o que o código faz, não como ele faz (o código já mostra “como”).
- Evite exagerar: bons comentários são curtos, claros e relevantes.
- Coloque lembretes de alguma melhoria futura possível

VARIÁVEIS

Objetivos de aprendizagem

- Compreender o conceito de variável e sua função na programação.
- Identificar e declarar variáveis corretamente em Python.
- Utilizar variáveis para armazenar e manipular dados de diferentes tipos.

Uma **variável** é um espaço nomeado na memória do computador usado para armazenar valores que podem ser modificados durante a execução de um algoritmo. Em Python, as variáveis são declaradas no momento da atribuição de valor, sem necessidade de declarar tipo previamente, pois a linguagem é dinamicamente tipada.

Atenção: a forma como uma variável é criada, possui um tipo de dado definido a ela e recebe valores por atribuição, depende de cada linguagem de programação. Algumas linguagens exigem que a variável seja criada, e seu tipo informado antecipadamente, antes de sua utilização.

* Importante: Em Python, não é necessário informar o tipo da variável ao criá-la. O tipo é definido automaticamente de acordo com o valor atribuído.

Exemplo:

```
nome = "João"
idade = 25
altura = 1.75
ativo=True
```

As variáveis acima armazenam, respectivamente, uma string, um inteiro, um número real (com casas decimais) e um valor booleano. Estudaremos cada um desses tipos em seguida.

Exemplo de uma atribuição de valores dinâmica em Python

```
x = 10
print(x)
x = "dez"
print(x)
```

Uma variável pode trocar de valor e de tipo em tempo de execução. Isso é chamado de tipagem dinâmica.

Atenção: podemos fazer uma comparação de uma variável com uma caixinha, ou uma gaveta, com uma etiqueta (nome) que guarda um valor dentro. Você pode abrir, ver, trocar o valor dentro da caixinha/gaveta sempre que quiser durante o programa

.

Os nomes de variáveis devem seguir boas práticas:

- Ser significativos, ou seja, seu nome deve dar uma ideia do seu conteúdo, como: nome_cliente, valor_venda, código_produto, etc.
- Não conter espaços em branco ou caracteres especiais;
- Não começar com números;
- Não usar palavras que são de uso específico da linguagem.

Atenção: um tipo especial de variável utilizada em programação é a CONSTANTE. Como o próprio nome diz, é um tipo de variável cujo valor não é alterado ao longo da execução do algoritmo. Um exemplo muito utilizado em cálculo é o valor de Pi (π), cujo valor é constante, não alterando durante a execução.

TIPOS DE DADOS

Objetivos de Aprendizagem

- Reconhecer os tipos de dados primitivos utilizados em Python.
- Identificar a aplicação correta de cada tipo de dado conforme o contexto.
- Utilizar os tipos de dados para entrada, armazenamento e manipulação de informações.

Em programação, **os tipos de dados** determinam o formato dos valores que podem ser armazenados em variáveis. No Python, os principais tipos primitivos são:

int	25	Números inteiros, positivos ou negativos
float	3.14	Números reais com ponto decimal
str	"Ana"	Cadeias de caracteres (textos)
bool	True, False	Valores lógicos: verdadeiro ou falso

Atenção
O Python utiliza o ponto final “ “ para separador decimal.
Strings são delimitadas por aspas simples ou duplas.
Os tipos são inferidos automaticamente com base no valor atribuído.

Atenção: atentar para a diferença entre valores que representam string (caracteres) e números. Veja os exemplos abaixo e os respectivos resultados.

```
numero = 25        # tipo int
texto = "25"       # tipo str
print(numero + 1)  # funciona
print(texto + 1)     # erro, pois não podemos realizar cálculos entre variáveis str e numéricas (int ou float)
```

✱Importante: em Python existe uma função muito útil para identificarmos o tipo de dados de cada variável. É a função type(). Veja os exemplos abaixo:

```
idade = "30”
print(type(idade))  # <class 'str'>

idade = 30
print(type(idade))  # <class 'int”'>
```

OPERADORES E EXPRESSÕES

Objetivos de Aprendizagem

- Identificar e aplicar operadores aritméticos, relacionais e lógicos em Python.
- Construir expressões matemáticas e lógicas corretamente.
- Compreender o funcionamento da precedência de operadores.

Conceito Teórico

Inicialmente vamos entender o conceito de **Expressão.**

Expressões são combinações de valores, variáveis e operadores que podem ser avaliadas para produzirem um resultado.

Exemplo de uso de expressões:

```
total = 10 + 5 * 2
print(total)  # Resultado: 20
```

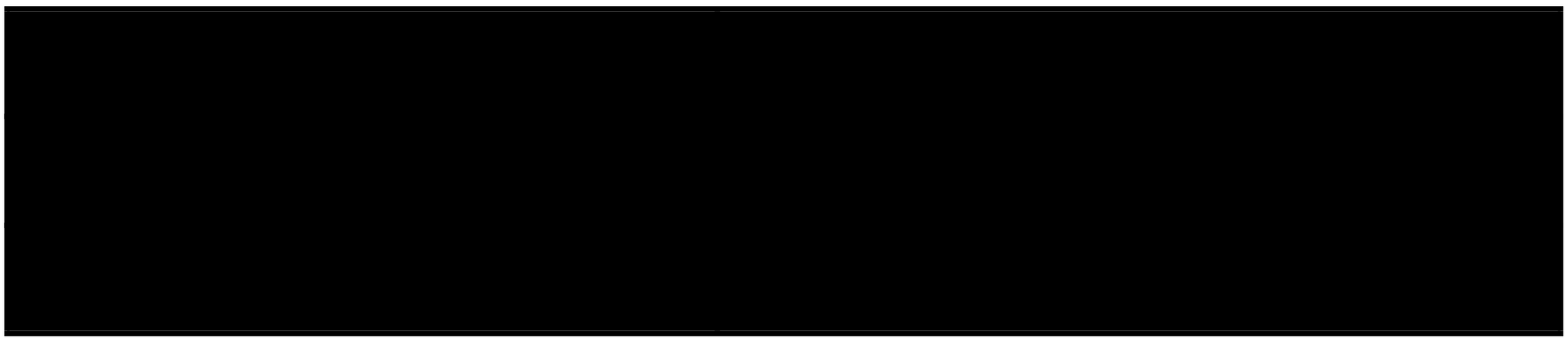
Atenção: além de observarmos o uso de uma expressão para realizar um cálculo, neste exemplo percebemos o uso de precedência dos operadores matemáticos, com a multiplicação primeiro, depois a adição.

Em programação, operadores são símbolos utilizados para realizar operações sobre variáveis e valores.

Operadores Aritméticos

Operadores Relacionais

Usados para comparar valores. Retornam True ou False.



Exemplo do uso de operadores relacionais, comparando dois valores, cujo resultado da avaliação, True ou False, é apresentado na tela pela função print.

```
idade = 18
print(idade >= 18) # True
print(idade < 18) # False
```

Operadores Lógicos

Permitem compor condições mais complexas:

Exemplo de uso dos operadores lógicos para validação de uma condicional

```
senha_correta = True
cpf_valido = False
print(senha_correta and cpf_valido)  # False
```

Atenção: no exemplo acima, as variáveis recebem por atribuição valores booleanos, ou seja, valores que representam, respectivamente, verdadeiro e falso.

Quadro resumo dos operadores

Objetivos de aprendizagem

- Utilizar as funções input( ) e print( ) em Python para interagir com o usuário.
- Realizar leitura e exibição de dados de forma clara e adequada.
- Combinar dados e mensagens em saídas formatadas.

Conceito Teórico

A interação com o usuário é essencial na construção de algoritmos. Em Python, usamos:

Entrada de Dados – input( )

Permite capturar dados digitados pelo usuário. O valor retornado é sempre do tipo str (string), sendo necessário conversão para outros tipos conforme o uso.

Para refletir: por que precisamos converter tipos de dados? Vamos ver o exemplo abaixo que apresenta o impacto da falta da conversão do tipo de dado correto para as operações desejadas.

```
idade = input(\"Digite sua idade: \")
print(idade + 1)  # Erro! idade é string
```

Solução, com conversão de tipo de dado. Nesse caso utilizamos a função int para converter de str para int

```
idade = int(input(\"Digite sua idade: \"))
print(idade + 1)
```

Praticando:

* No exemplo abaixo, um está incompleto. Execute e descubra qual, corrigindo-o.

```
nome = input("Digite seu nome: ")
salario = float(“Digite o valor do salário: “) idade
= int(input("Digite sua idade: "))
```

🖥 Saída de Dados – print( )

Exibe mensagens, valores e resultados formatados na tela.
Pode concatenar strings, imprimir múltiplos valores ou usar formatação com f-strings.

❯ 01. **Uso Básico de print( )**

A função print( ) serve para exibir mensagens ou valores no terminal.

```
print("Olá, mundo!")
```

❯ 02. **Imprimindo Múltiplos Valores**

Podemos imprimir diversos elementos separados por vírgulas.
O Python automaticamente adiciona espaços entre os itens.

```
nome = "Carlos"
idade = 30
print("Nome:", nome, "- Idade:", idade)
```

Saída:

```
Nome: Carlos - Idade: 30
```

❯ 03. **Concatenação de Strings com +**

Para juntar textos, podemos usar o operador +. Todos os itens concatenados devem ser str (strings).

```
nome = "Ana"
print("Bem-vinda, " + nome + "!")
```

🧠*Para refletir: em Python, concatenar string com outros tipos (como int) gera erro.*
🧪*Pratique concatenação entre diferentes tipos de dados e avalie os motivos do erro.*

```
idade = 20
print("Idade: " + str(idade)) # Correto
```

❯ 04. **Uso de f-strings (Formatação moderna)**

As **f-strings** são uma forma prática e moderna de incluir variáveis diretamente no texto.

```
nome = "João" nota
= 8.75
print(f"O aluno {nome} obteve nota {nota}")
```

Formatação com casas decimais:

```
valor = 123.4567
print(f"Valor formatado: {valor:.2f}")
```

❱ 05. **Quebra de Linha e Caracteres Especiais**

O \n insere uma **quebra de linha:**

```
print("Linha 1\nLinha 2")
```

Outros caracteres:

- \t: tabulação
- \\: barra invertida
- \" ou \': aspas dentro de strings

❱ 06. **Mudança de Final (end)**

O argumento end define o que será colocado no fim da linha (por padrão é \n).

```
print("Item 1", end=" | ")
print("Item 2", end=" |")
print("Item 3")
```

Saída:

```
Item 1 | Item 2 | Item 3
```

❱ 07. **Separador de Valores (sep)**

O argumento sep define o separador entre múltiplos argumentos no print().

```
print("10", "20", "30", sep=" - ")
```

Saída:

```
10 - 20 - 30
```

❱ 08. **Print com Expressões**

```
a = 5
b = 3
print("Soma:", a + b)
```

❱ 09. **Uso com Strings de Múltiplas Linhas**

```
print("""
Informações:
- Nome: João
- Curso: ADS
- Ano: 2024
""")
```

Resumo das Boas Práticas

- Use vírgulas para imprimir diferentes tipos sem erro de tipo.
- Para saídas formatadas, prefira f-strings.
- Sempre converta tipos antes de concatenar com +.
- Evite usar print() para lógica ou cálculos complexos – mantenha-o para exibição.

Objetivo

Mostrar como usamos lógica no dia a dia, antes mesmo de programar, reforçando que os algoritmos são a tradução dessas decisões para o computador.

Lógica na Vida Real

No cotidiano, tomamos decisões com base em condições, sem perceber:

- **Exemplo simples:** “Se hoje é dia útil (condição), então levanto cedo (ação); caso contrário, durmo até mais tarde.”
- **Motivação:** Mostrar que no nosso cotidiano utilizamos a lógica para analisar situações e tomar decisões, e nossos passos durante o dia são um tipo de estrutura de algoritmos, com decisões estruturadas por condições lógicas.

E, para colocar ordem no pensamento e na execução das nossas ações, necessariamente e diretamente, usamos a lógica. Então, entenda com o exemplo a seguir:

1. Paulo mora na cidade de Florianópolis
2. Silvia mora na mesma cidade que Paulo
3. Então, Silvia mora em Florianópolis

Ainda:

1. Carlos foi campeão na prova do Iron Man 2025
2. André participou da mesma prova que Carlos
3. Portanto, André NÃO foi o campeão da prova

Portanto, é possível afirmar que a lógica analisa e confirma as proposições, definindo sua veracidade (verdade) ou não (falso) e nos permite tomar decisões a partir dos resultados. Ou seja, chegar à determinada conclusão e, assim, decidirmos qual ação realizar baseados no resultado.

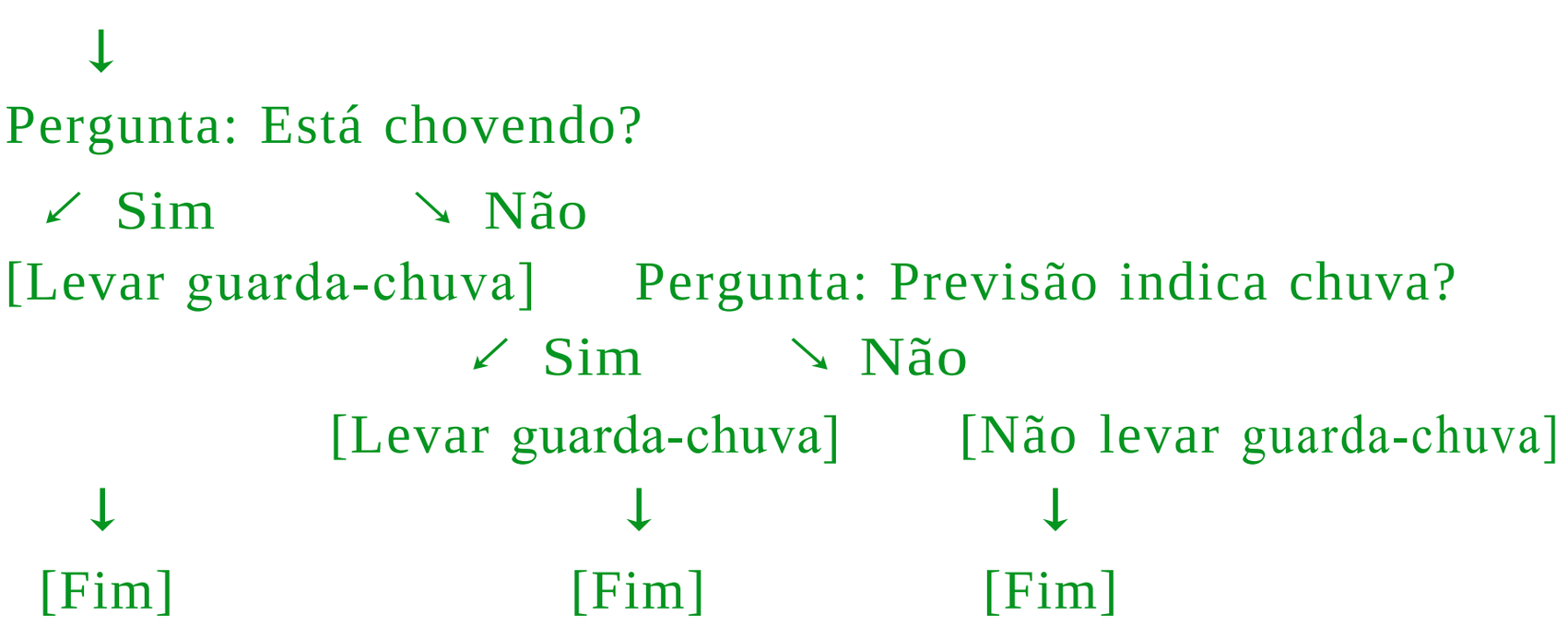
Exemplo Prático: Decisão sobre Levar o Guarda-Chuva

Situação

Você sai de casa e observa:

- Se estiver chovendo → leva guarda-chuva.
- Se não estiver chovendo mas a previsão indica chuva → leva guarda-chuva.
- Caso contrário → não leva.

Fluxograma



Versão em Python

Todos os comandos deste algoritmo serão estudados à medida que avançarmos nos tópicos seguintes.

```
chove = True # variável do tipo booleano - True ou False
previsao_chuva = False # variável do tipo booleano - True ou False if
chove:
  print("Levar guarda-chuva (chovendo)") else:
  if previsao_chuva:
    print("Levar guarda-chuva (choverá)") else:
    print("Não levar guarda-chuva (sem chuva)")
print("FIM")
```

NIVELAMENTO

LÓGICA COMPUTACIONAL

A lógica computacional é aplicada para ordenar o pensamento e expressá-lo por meio de comandos que o computador consegue interpretar. Essa organização permite criar rotinas capazes de automatizar processos e resolver problemas diversos, desde os mais simples até os mais complexos.

Já vimos que a lógica está presente no dia a dia das nossas ações e decisões. Seguindo essa linha de raciocínio, pode-se concluir que a lógica de programação é o **uso de técnicas para encadear ações e pensamentos, com a finalidade de atingir determinado objetivo por meio das proposições e das condicionais (situações verificadas e avaliadas), elaboradas em um fluxo e na sequência de determinados e finitos passos.**

As linguagens de programação possuem comandos que permitem a um algoritmo analisar situações (condicionais) e, baseado na resposta, seguir determinado fluxo de instruções, ou comandos, agrupados em blocos de código.

Esses comandos, juntamente com as condicionais e os blocos de código são chamados de Estruturas condicionais, que estudaremos a seguir.

UniSENAI

Objetivos de Aprendizagem

- Compreender a estrutura if, elif, else em Python e sua relação com proposições lógicas.
- Desenvolver a capacidade de análise lógica por meio de inferência e tomada de decisão.
- Aplicar estruturas condicionais em algoritmos para resolver problemas reais com clareza de lógica.

Conceito Teórico

A **estrutura condicional** é usada para **tomar decisões**, executando blocos diferentes de código dependendo do resultado de **expressões lógicas** (proposições).

Vimos essa ideia no exemplo da decisão a ser tomada, levar ou não um guarda-chuva, baseados na resposta sobre possibilidade de chuva. Pois, em nosso cotidiano tomamos decisões com base em condições: se estiver chovendo, levo guarda-chuva. Se não estiver, saio sem ele. Programas de computador fazem o mesmo.

Proposição

Uma proposição é uma **afirmação que pode ser verdadeira ou falsa**, nunca ambos ao mesmo tempo.

Por exemplo, uma pessoa não pode ser menor e maior de idade ao mesmo tempo, um número não pode ser par e ímpar, positivo e negativo.

```
idade = 18
if idade >= 18:
    print("Maior de idade")
else:
    print("Menor de idade")
```

Aqui, a proposição **idade >= 18** será avaliada e, com base na lógica, uma das ações será tomada.

Recordando que “idade” é uma variável que possui um valor inteiro armazenado. Se esse valor for maior ou igual a 18, então a condicional será verdadeira e a mensagem “Maior de idade” será mostrada na tela. Caso o

valor armazenado seja menor que 18, a mensagem “Menor de idade” é apresentada na tela.

✱Importante: notem no exemplo acima que as linhas com as funções print() estão deslocadas para a direita. Essa é uma especificidade do Python, que identifica um bloco de código através do deslocamento da linha para a direita, com o uso da tecla Tab ou com 4 espaços em branco. Isso é chamado de indentação. Esse conceito é utilizado em bloco de código das condicionais (if/elif/else) e para os comandos de controle de repetição, que estudaremos no tópico.II

Encadeamento com elif

Permite testar múltiplas proposições:

```
if nota >= 6:
    print("Aprovado")
elif nota >= 3:
    print("Em exame")
else:
    print("Reprovado")
```

Exemplo com o uso progressivo da estrutura if/elif/else para as três possibilidades de resultado como condição do aluno em unidades curriculares do UniSenai.

```
nota = float(input("Digite sua nota: "))
if nota >= 6:
    print("Aprovado")
elif nota >= 3:
    print("Em Exame")
else:
    print("Reprovado")
```

Pensamento Lógico

A construção de algoritmos com condicionais exige **antecipar situações, avaliar resultados e condições e organizar o fluxo de decisão, considerando as possibilidades de resposta: verdadeiro ou falso.**

Pensar como um computador significa transformar regras do cotidiano em condições exatas, com clareza e sem ambiguidades. Em lógica, a certeza é baseada na possibilidade de resposta, verdadeiro ou falso. Não existe a incerteza do “talvez”, “quem sabe”, “depende”, ou outra condição que não seja exata.

LÓGICA PROPOSICIONAL E DESVIOS CONDICIONAIS

Objetivos Complementares

- Compreender a lógica formal por trás dos desvios condicionais em algoritmos.
- Analisar a tabela-verdade para proposições simples e compostas.
- Aplicar esse conhecimento para construir algoritmos com decisões encadeadas.

Nossos estudos agora focam em revisarmos os operadores relacionais e lógicos e seu uso na lógica das proposições.

Lembramos e destacamos que, ao programar, tomamos decisões baseadas em condições. Essas condições são expressões lógicas. Por trás delas, existe uma base matemática chamada lógica proposicional.

Então, uma proposição é uma afirmação, que ao ser validada (verificada), retorna um resultado como verdade (True) ou falso (False).

Exemplos do cotidiano:

“Hoje é segunda-feira”: ao avaliarmos a afirmação, teremos como resposta verdadeiro ou falso.

“Feche a porta”: pode ser visto como um comando, mas não representa uma proposição, pois não tem sentido, nem é possível, realizar uma avaliação.

5 > 3: estamos afirmando que cinco é maior que três, então será feita uma validação.

media == 9: estamos afirmando que a média é igual a nove, então será feita uma validação, recuperando o valor armazenado na variável “media” e verificando se o mesmo é igual a nove.

Os **Operadores Relacionais** são utilizados para comparar valores e o resultado dessa comparação sempre será verdade ou falso.

Ou seja, retornam como resultado True ou False.

E os **operadores lógicos** (conectivos lógicos) permitem compor condições mais complexas, onde a proposição lógica avaliada possui mais de uma condição (comparação de valores). Então, um operador lógico realiza a concatenação entre duas condicionais.

LÓGICA DE PROPOSIÇÕES E TABELA-VERDADE

Proposição Simples

Uma proposição simples é uma afirmação que pode ser Verdadeira (V) ou Falsa (F) baseada em **uma única condicional**.

```
chove = True # variável do tipo booleano - True ou False
if chove:
  print("Levar guarda-chuva (chovendo)")
else:
  print("Não levar guarda-chuva (sem chuva)")
print("FIM")
```

Conectivos Lógicos

“A tabela verdade é o conjunto de todas as possibilidades combinatórias entre os valores de diversas variáveis lógicas, as quais se encontram em apenas duas situações (Verdadeiro ou Falso), e um conjunto de operadores lógicos”. (FORBELLONE, EBERSPÄCHER, 2005).

Quando a condicional é formada por uma única condição, não utilizamos a tabela verdade, pois teremos somente UMA avaliação a ser realizada.
Por exemplo:

```
if x < 0: # SE um número é negativo
if nota > 7: # SE uma nota é maior que 7
if idade >= 18: # SE uma pessoa é maior de idade
if temperatura < 0: # SE uma temperatura é negativa
```

Porém, muitas vezes temos a necessidade de realizarmos duas ou mais condicionais, duas ou mais avaliações lógicas. E nesse caso, precisamos conhecer e saber utilizar a tabela verdade e o uso dos operadores lógicos.

Vamos apresentar a tabela verdade dos operadores lógicos AND, OR e NOT, com exemplos.

Tabela-Verdade:
Operador AND (and)

Considerações: uma avaliação com o operador AND entre duas condicionais será VERDADEIRA **somente se ambas resultarem em verdadeiro.** Caso contrário, será FALSO.

Exemplo: o aluno será aprovado se a média for maior ou igual a seis **E** se tiver no mínimo 75% de presença:

Código: **if media >= 6.0 and frequência >= 75:**

Operador OR (or)

“Considerações: uma avaliação com o operador OR entre duas condicionais será VERDADEIRA sempre que **pelo menos uma delas resultar em verdadeiro.** Se ambas forem falso, a resposta final será FALSO.

Exemplo: o aluno não estará em Avaliação Complementar SE, estiver Aprovado (a média é maior ou igual a 6) ou SE estiver reprovado (a média é menor que 3).

Código: **if media < 3.0 or media >= 6.0**:

Operador NOT (not)

Atenção: o operador NOT, nos permite avaliar ao negar a resposta de uma condicional. Se o resultado for verdadeiro, o operador NOT gera um resultado falso. O mesmo para o contrário, se o resultado for falso, NOT gera verdadeiro.

Exemplo: avaliar se um número é diferente de zero.
Temos duas opções:

(1) avaliar se um número é diferente de zero:
Código: **if numero != 0**

(2) negar se um número é igual a zero.
Código: **if not numero == 0**

Proposição Composta

Uma proposição composta é uma afirmação que pode ser Verdadeira (V) ou Falsa (F) baseada em duas condicionais.

Cada condicional é avaliada e depois cada resultado é analisado baseado na tabela verdade do operador lógico da proposição. O resultado dependerá dos resultados obtidos e da combinação deles na respectiva tabela verdade.

Abaixo o exemplo da decisão em levar, ou não, o guarda-chuva, baseado nas avaliações lógicas, desta vez utilizando operadores lógicos para conectar duas proposições.

```
chove = True # variável do tipo booleano - True ou False
tenho_guarda_chuva = True # variável do tipo booleano - True ou False
previsao_chuva = False # variável do tipo booleano - True ou False
if chove and tenho_guarda_chuva:
print("Levar guarda-chuva (chovendo)")
else:
  print("Melhor não sair de casa")
if chove or previsao_chuva:
  print("Levar guarda-chuva (chove ou choverá)")
else:
  print("Vou acaber me molhando")
print("FIM")
```

Desvios Condicionais

Desvio condicional” é quando o programa toma uma decisão sobre qual caminho seguir com base no resultado de uma condição. A execução não segue linha a linha: ela desvia dependendo de True ou False.

Já estudamos a estrutura if/elif/else, utilizada para os desvios condicionais, conforme exemplos abaixo.

Antes, vamos revisar os elementos básicos da estrutura condicional

Simples (if)

Nem todo desvio condicional precisa necessariamente de outro caminho, exceto aquele que é executado caso a condição for verdadeira

Exemplo1:

```
num = 10
if num % 2 == 0:
    print("Número PAR")
```

Exemplo2:

```
temperatura = - 4
if temperatura < 0 :
    print(“Temperatura negativa detectada”)
```

Com else

Exemplo 1:

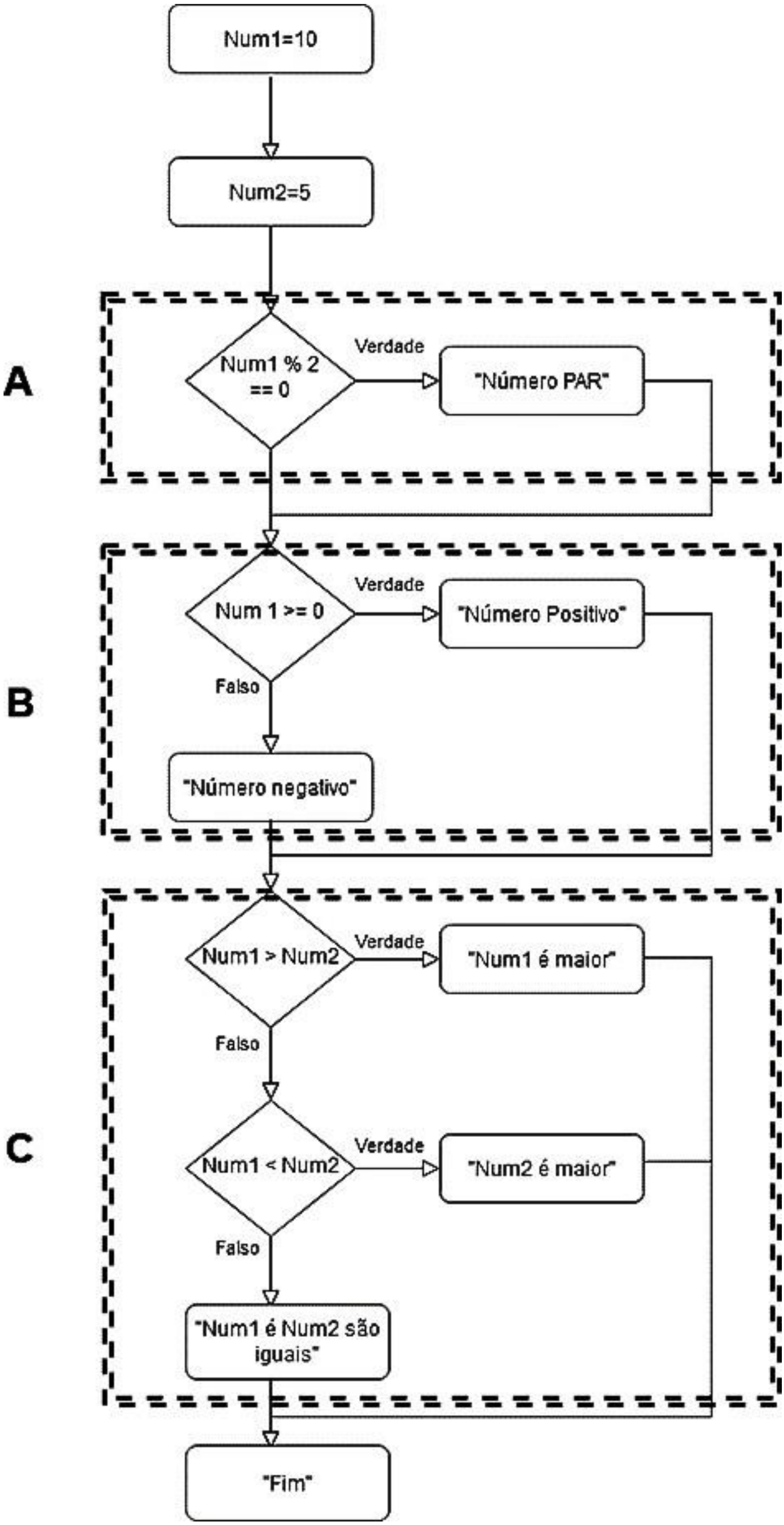
```
num = 13
if num > 0:
    print("Número Positivo")
else:
    print("Número Negativo")
```

Múltiplos caminhos (if/elif/else)

Exemplo 1:

```
if num1 > num2:
    print("num1 é maior")
elif num1 < num2:
    print("num2 é maior")
else:
    print("num1 e num2 são iguais")
```

Observe fluxograma abaixo. Ele descreve os desvios condicionais possíveis de ocorrer em uma proposição lógica.


Num1=10
Num2=5
A
Num1 % 2 == 0
Verdade
"Número PAR"
B
Num 1 >= 0
Verdade
"Número Positivo"
Falso
"Número negativo"
C
Num1 > Num2
Verdade
"Num1 é maior"
Falso
Num1 < Num2
Verdade
"Num2 é maior"
Falso
"Num1 é Num2 são iguais"
"Fim"

1) Classificação de variáveis

Indique o tipo de dado apropriado para as variáveis:

- temperatura: int
- nome_cliente: string
- saldo_bancario: float
- idade: int
- altura: float
- ativo: bolean (para verdadeiro/falso)

2) Preenchendo variáveis

Complete o código abaixo com valores compatíveis:

Tipo = Locadora de Filme
produto = YoungHeart
quantidade = 1
preco_unitario = 27.5
disponivel = True

3) Teste de mesa

Analise:

```
a = 10
b = 2
c = a + b
print(c)
b = 5
c = a * b
print(c)
```

Quais os valores serão apresentados na tela?

Int

4) Qual das alternativas representa corretamente a criação de variáveis em Python?

a) int idade = 25
b) nome: str = "Ana"
c) idade = 25
d) altura == 1.65
e) def nome = "Carlos"

5) Identifique o tipo

Associe cada valor ao tipo de dado correspondente:

- "45.5": string
- False: Bolean
- 100: int
- "Maria": string
- 3.1415: float

6) Análise de código

O que será exibido na tela?

idade = "30"

print(idade + 5)

R: TypeErro

7) Explique o que ocorre em cada linha:

a = "5" - chamou a variavel a atribuindo a string 5 a ela

b = int(a) – chamou a variavel b atribuindo o valor da variavel a convertido para int

c = float(b) – Converteu o valor de b para float e atribuiu a variavel c

d = str(c) - Converteu o valor de c para string e atribuiu a variavel d

8) Em Python, qual das afirmações está correta sobre tipos de dados?

a) O tipo float serve para textos curtos.

b) O tipo bool pode armazenar números reais.

c) O valor True é o valor booleano equivalente ao resultado verdade de uma condição.

d) A string "25" pode ser usada diretamente em cálculos aritméticos.

e) O tipo int é usado para textos numéricos.

9) Complete a tabela:

Avalie o resultado das expressões com a = 5, b = 2:

Atenção: observe o tipo de operador na expressão para responder

Resultado da tabela: a+b = 7, a%b = 1, a > b = true, a === 5 and b < 3 = verdadeiro, not(a===b) = verdadeiro

Expressão	V]
a + b	
a % b	
a > b	
a == 5 and b < 3	
not(a == b)	

10) Analise e explique o resultado:

x = 10

y = 4

print((x % 3) + (y // 2))

R: O resultado é 3 pq o resto 1 somando a divisão de 10 por 4 mas só contando o inteiro é 3 vindo de 1+2

11) Precedência de operadores:

Qual o resultado das operações matemáticas abaixo?

a) 3 + 4 * 2 = 11

b) 3 + (4 * 2) = 11

12) Considere o seguinte código:

x = 7

y = 3

resultado = x % y == 1 and x > y

Qual é o valor de resultado?

a) 1

b) True

c) False

d) None

e) Erro de sintaxe

13) Identifique a saída esperada:

produto = "Caderno"

preco = 14.75

print("Produto:", produto, "- Preço: R$", preco)

R? Produto: Caderno – Preço R$ 14.75

14) Escreva o código que atende o enunciado:

Faça com que o programa leia o nome de um usuário e a cidade onde mora, e imprima: Fulano mora em CidadeX.

nome = input(“Digite seu nome: ”)

cidade = input(“Digite sua cidade: ”)

print(f”{nome} mora em {cidade}”)

15) Correção de erro

Analise e corrija o erro:

```
idade = input("Digite sua idade: ")
print("Você terá", idade + 1, "anos no próximo ano.")
```

Correção:

```
idade = int(input("Digite sua idade: "))
print("Você terá", idade + 1, "anos no próximo ano.")
```

16) Qual das opções abaixo imprime corretamente o nome de um usuário lido pelo teclado, junto com uma mensagem personalizada?

a)
```
nome = input("Nome")
print("Olá" + nome)
```

b)
```
nome = input("Digite seu nome: ")
print("Olá,", nome)
```

c)
```
input(nome)
print(nome + "Olá")
```

d)
```
print("Digite seu nome:", input(nome))
```

e)
```
nome = input
print("Olá", nome)
```

17) Analise as proposições abaixo e classifique como Verdadeira ou Falsa, considerando x = 5, y = 10:

- *x > y*
- *x <= 5*
- *not(x == y)*
- *x * 2 == y*
- *x == 5 and y == 10*

18) Complete a sequência lógica de decisão escrevendo a lógica das condicionais abaixo em Python:

Se a idade for maior ou igual a 60 → Idoso
Se for maior ou igual a 18 → Adulto
Caso contrário → Menor de idade

19) Explique o problema no uso do operador no código abaixo, considerando que o erro é na condicional, não se tratando de um erro de execução do algoritmo:

```
if nota < 6:
    print("Aprovado")
else:
    print("Reprovado")
```

20) Considere:

```
x = 10
y = 20
if x < y and x % 2 == 0:
    print("Condição verdadeira")
else:
    print("Condição falsa")
```

Qual será a saída?

a) Erro de sintaxe
b) Condição verdadeira
c) Condição falsa
d) 10
e) Nenhuma

UniSENAI