



Linguagem de Programação

Python - Básico

Bem-vindo ao mundo fascinante da programação Python!

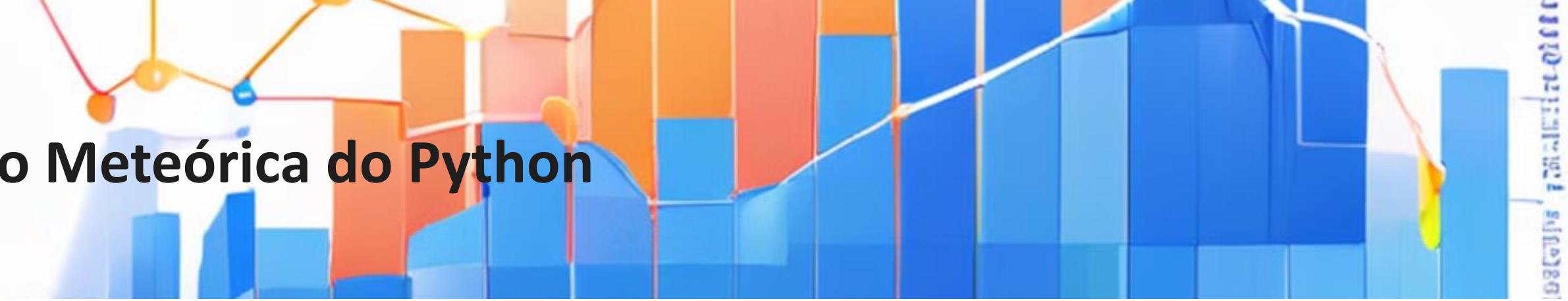
Esta linguagem versátil e poderosa tem conquistado corações de programadores iniciantes e experientes em todo o mundo.

Com sua sintaxe clara e intuitiva, Python oferece uma porta de entrada acolhedora para quem está começando a explorar o universo da programação.

Ao mesmo tempo, sua robustez e flexibilidade a tornam uma escolha popular entre profissionais em campos diversos, desde desenvolvimento web até inteligência artificial.

Prepare-se para embarcar em uma jornada de aprendizado empolgante e descobrir por que Python é considerada uma das linguagens mais amadas e utilizadas globalmente.

A Ascensão Meteórica do Python



Criação e Lançamento

Python foi criado por Guido van Rossum e lançado pela primeira vez em 1991, com foco em legibilidade e simplicidade.

Crescimento Constante

Ao longo dos anos 2000 e 2010, Python ganhou popularidade steadily, especialmente em comunidades científicas e acadêmicas.

Explosão de Popularidade

Nos últimos anos, Python experimentou um crescimento exponencial, impulsionado pelo boom de data science e machine learning.

<https://www.tiobe.com/tiobe-index/>

Versatilidade Incomparável

Desenvolvimento Web

Com frameworks como Django e Flask, Python permite criar aplicações web robustas e escaláveis de forma eficiente.

Ciência de Dados

Bibliotecas como NumPy, Pandas e Matplotlib fazem de Python a escolha número um para análise e visualização de dados.

Inteligência Artificial

TensorFlow e PyTorch, duas das principais bibliotecas de machine learning, são baseadas em Python, tornando-a essencial para IA.

Simplicidade e Legibilidade

Sintaxe Clara

Python utiliza indentação para definir blocos de código, resultando em uma estrutura visualmente limpa e fácil de entender.

Menos é Mais

Com menos símbolos e palavras-chave, Python reduz a complexidade visual do código, tornando-o mais acessível para iniciantes.

Pseudocódigo Executável

Muitos descrevem Python como "pseudocódigo executável", pois sua sintaxe se assemelha à descrição lógica de algoritmos em linguagem natural.

Curva de Aprendizado Suave

A simplicidade de Python permite que novos programadores se concentrem em conceitos de programação, em vez de peculiaridades da linguagem.

Comunidade Vibrante e Recursos Abundantes

1 Fóruns e Grupos de Discussão

Plataformas como Stack Overflow e Reddit hospedam comunidades Python ativas, onde programadores de todos os níveis podem trocar conhecimentos e resolver problemas.

2 Documentação Abrangente

A documentação oficial do Python é conhecida por sua clareza e completude, oferecendo um recurso valioso para aprendizado e referência.

3 Conferências e Eventos

Eventos como PyCon e Django Con reúnem entusiastas de Python de todo o mundo, promovendo networking e compartilhamento de conhecimento.

4 Bibliotecas de Código Aberto

O Python Package Index (PyPI) hospeda mais de 300.000 pacotes de software, fornecendo soluções para praticamente qualquer necessidade de programação.





Python no Mercado de Trabalho

| Área | Demanda | Salário Médio (BR) |
|---------------------|------------|------------------------|
| Desenvolvimento Web | Alta | R\$ 5.000 - R\$ 12.000 |
| Ciência de Dados | Muito Alta | R\$ 6.000 - R\$ 18.000 |
| Machine Learning | Muito Alta | R\$ 8.000 - R\$ 20.000 |
| Automação | Média-Alta | R\$ 4.000 - R\$ 10.000 |



Conhecendo os Operadores Lógicos em Programação

Essenciais para qualquer programador, esses operadores são as ferramentas que nos permitem criar condições complexas e tomar decisões em nossos códigos.

Nesta apresentação, vamos explorar os três principais operadores lógicos: E (AND), OU (OR) e NAO (NOT).

O Operador Lógico E (AND)

Definição

O operador lógico E (AND) é fundamental na programação. Ele retorna verdadeiro apenas quando todas as condições avaliadas são verdadeiras.

Exemplo Prático

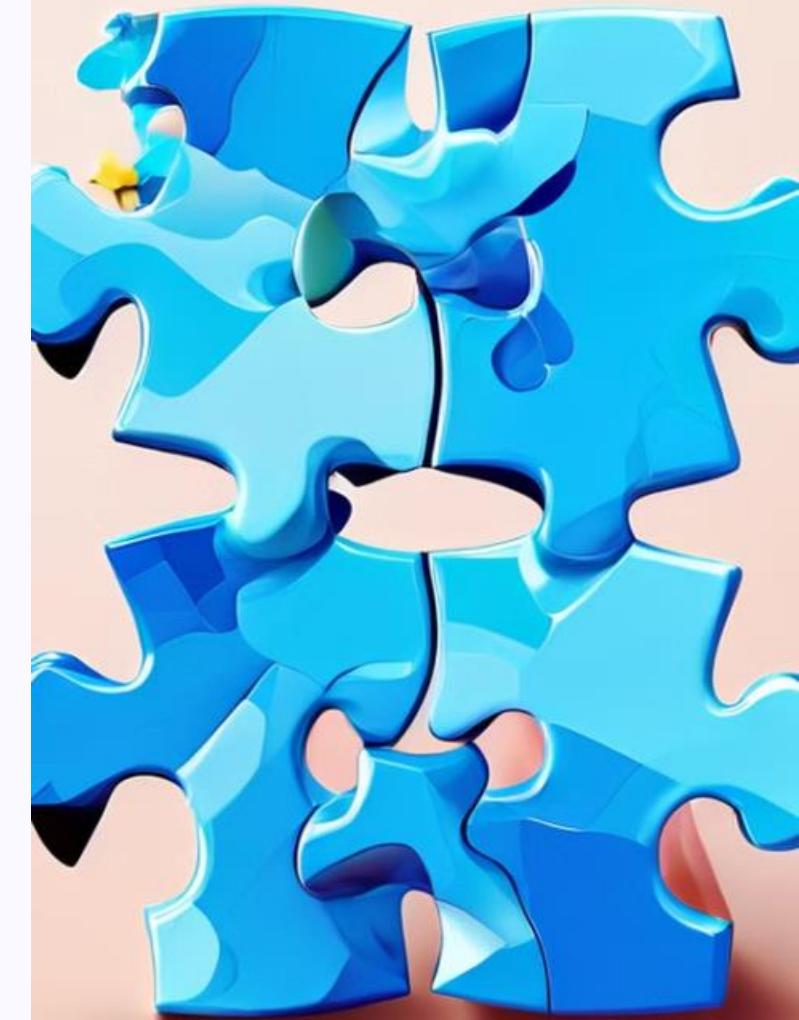
Em Python: `if (idade >= 18 AND tem_carteira == True): print("Pode dirigir"). dirigir").` Ambas as condições precisam ser verdadeiras para o código ser executado.

Funcionamento

Imagine o AND como um inspetor rigoroso: só aprova se todos os critérios forem atendidos. Se qualquer condição for falsa, o resultado será falso.

Importância

O AND é crucial para criar filtros precisos e garantir que múltiplas condições sejam satisfeitas antes de executar uma ação no programa.





O Operador Lógico OU (OR)

- 1 Compreendendo o OR

O operador OR é mais flexível que o AND. Ele retorna verdadeiro se pelo menos uma das condições for verdadeira. Apenas quando todas as condições são falsas, o resultado será falso.
- 2 Aplicação Prática

Por exemplo, em um sistema de desconto: `if (cliente_vip OR compra_acima_100): aplicar_desconto()`. O desconto `aplicar_desconto()`. O desconto será aplicado se qualquer uma das condições for verdadeira.
- 3 Versatilidade

O OR é incrivelmente útil para criar condições alternativas. Ele permite que seu código seja mais flexível, lidando com múltiplos cenários de uma só vez.
- 4 Cuidados ao Usar

Embora versátil, é importante usar o OR com cautela. Certifique-se de que a lógica do seu programa não se torne muito permissiva ao utilizar este operador.

O Operador Lógico NÃO (NOT)

1

Invertendo a Lógica

O operador NOT é único: ele inverte o valor da expressão. Se a condição for verdadeira, NOT a torna falsa, e vice-falsa, e vice-versa. É como um interruptor que inverte o estado atual.

2

Uso Prático

Em código:

```
if not (esta_chovendo):  
    print("Vamos ao parque!").
```

Aqui, o programa executa quando NÃO está chovendo, invertendo a lógica da condição.

3

Simplificando Condições

NOT pode simplificar condições complexas.

Por exemplo, "if not (x < 0 or x > 100)" é mais claro que
"if (x >= 0 and x <= 100)".

4

Cuidado com Dupla Negação

Ao usar NOT, evite duplas negações que podem confundir. "not (not verdadeiro)" é o mesmo que "verdadeiro", mas menos claro.

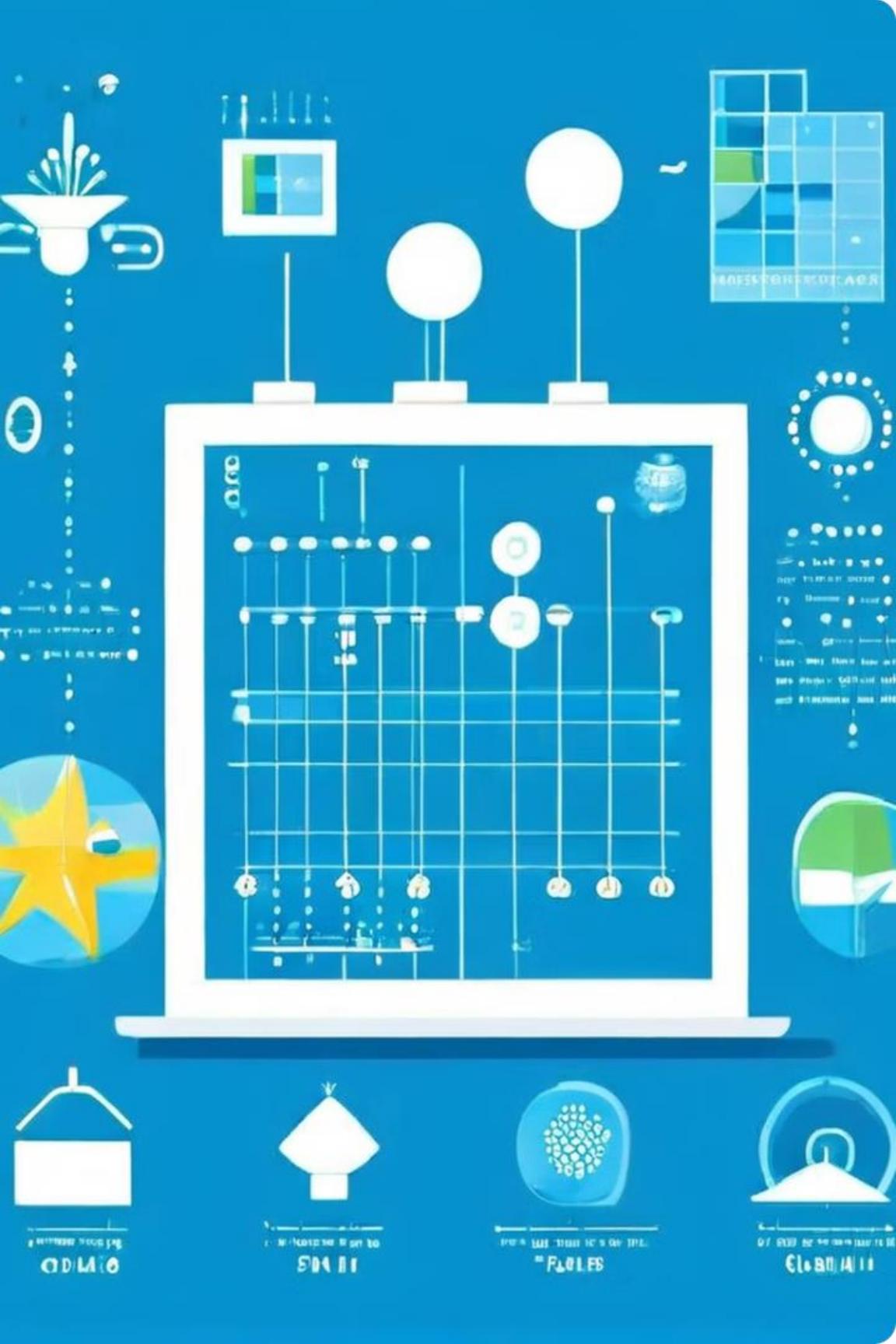


Tabela Verdade dos Operadores Lógicos Lógicos

| A | B | A AND B | A OR B | NOT A |
|---|---|---------|--------|-------|
| V | V | V | V | F |
| V | F | F | V | F |
| F | V | F | V | V |
| F | F | F | F | V |

A tabela verdade é uma ferramenta essencial para entender o comportamento dos operadores lógicos. Ela mostra todas as combinações possíveis de valores de entrada (verdadeiro ou falso) e os resultados correspondentes para cada operador.

Observe como o AND só resulta em verdadeiro quando ambas as entradas são verdadeiras, enquanto o OR é verdadeiro se pelo menos uma entrada for verdadeira. O NOT, por sua vez, sempre inverte o valor da entrada. Memorizar esta tabela ajudará você a prever o resultado de expressões lógicas complexas em seus programas.



Exercícios Práticos com Operadores Lógicos

Exercício 1: Operadores Relacionais

Considere $A = 5$ e $B = 3$. Determine se as seguintes expressões são verdadeiras ou falsas:

- $A > B$ AND $B < 4$
- $A == 5$ OR $B > 5$
- NOT $(A < B)$

Dica: Analise cada parte da expressão separadamente antes de aplicar os operadores lógicos.

Exercício 2: Combinando Operadores

Agora, com $A = 5$, $B = 8$ e $C = 1$, avalie avalie estas expressões:

- $(A > C)$ AND $(B < A)$
- $(B >= A)$ OR $(C != 1)$
- NOT $((A <= B)$ AND $(C < A))$

Lembre-se: A ordem de avaliação é importante. Resolva os parênteses internos primeiro!

Desafio Extra

Crie suas próprias expressões lógicas usando A , B e C . Tente combinar diferentes operadores e prever o resultado. Depois, verifique suas respostas usando um interpretador Python ou uma calculadora lógica online.

Praticar esses exercícios ajudará você a se tornar mais confiante no uso de operadores lógicos em seus programas!

Estrutura Lógica do Python

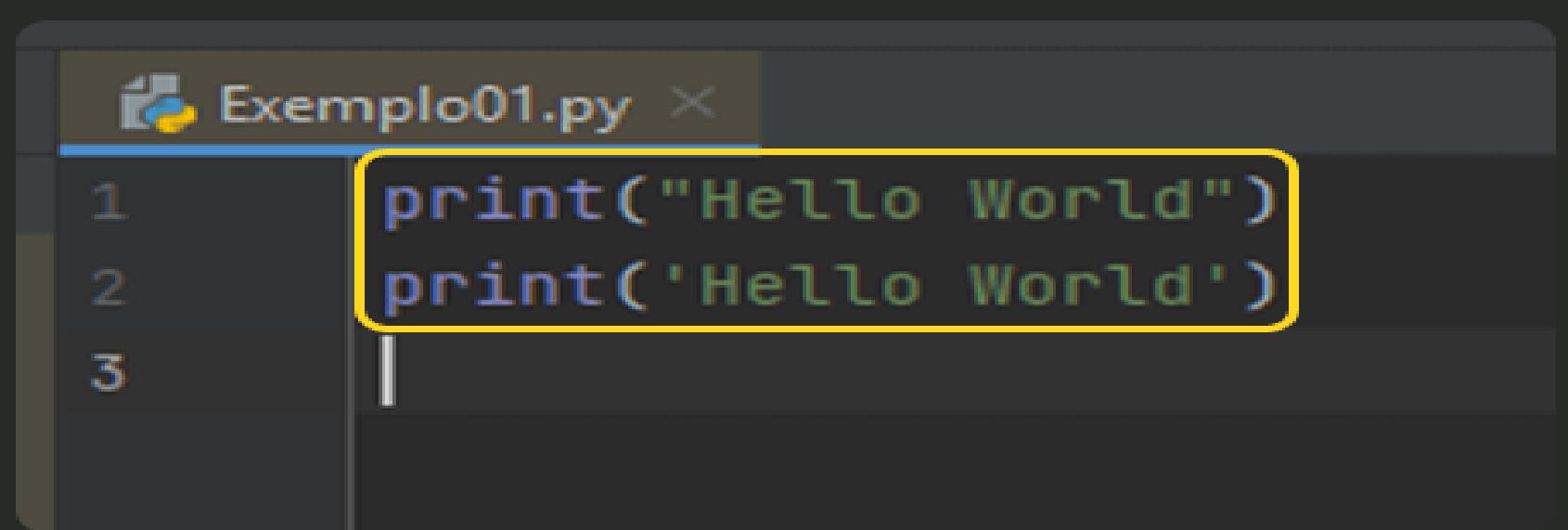
Vamos aprender sobre a estrutura lógica do Python, declaração de variáveis e tipos de dados.

Abordaremos comandos básicos, sintaxe e boas práticas de programação.



Comando print()

O comando print() é usado para imprimir mensagens de texto na tela. Sua sintaxe requer parênteses e aspas.



```
Exemplo01.py
1 print("Hello World")
2 print('Hello World')
```



Importante

Case Sensitive

A linguagem Python é *Case Sensitive*. Você sabe o que isso significa? Significa que ela faz **distinção entre letras maiúsculas e minúsculas**.

Portanto, é importante estar atento à escrita dos comandos. Por exemplo: a função **print** deve ser escrita em **letra minúscula**.





Erros Comuns

Erros de sintaxe podem ocorrer por uso incorreto de maiúsculas/minúsculas, falta de parênteses ou aspas.

1 Caixa Alta/Baixa

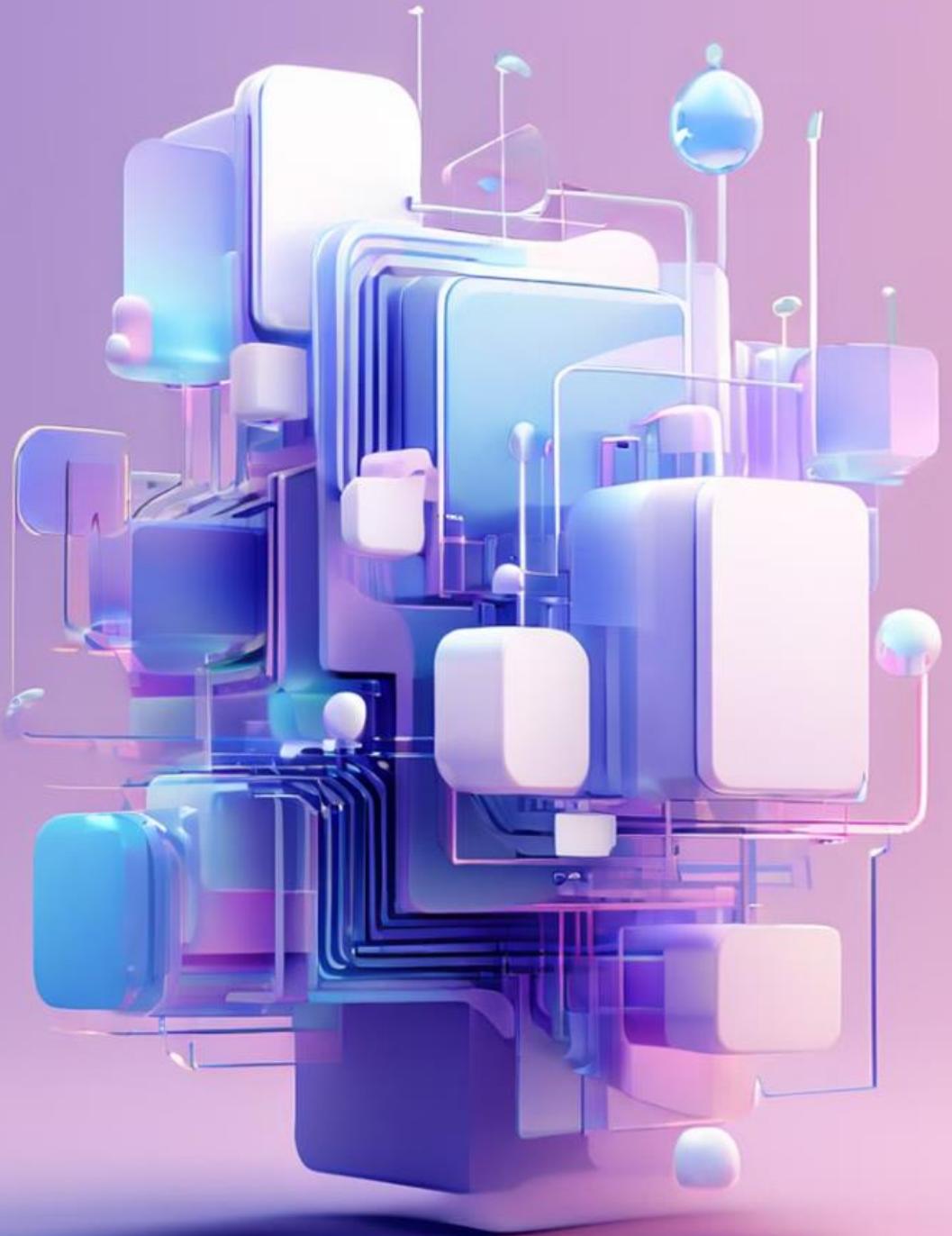
`Print ("Hello World")` - incorreto

2 Parênteses

`print (Hello World` - incorreto

3 Aspas

`print (Hello World")` - incorreto



Variáveis em Python

Variáveis em Python armazenam endereços de memória, não valores. Toda variável é uma referência a um objeto.



Memória

Armazena endereços



Objetos

Variáveis são referências



Dinâmico

Tipos podem mudar



Declaração de Variáveis

Variáveis podem ser declaradas a qualquer momento. Python interpreta automaticamente o tipo de dado.

| Declaração | 1 | |
|--|---|---|
| Pode ser feita em qualquer parte do código | | |
| Interpretação | 3 | 2 |
| Python reconhece o tipo automaticamente | | Não é necessário definir explicitamente |

Tipos de Variáveis

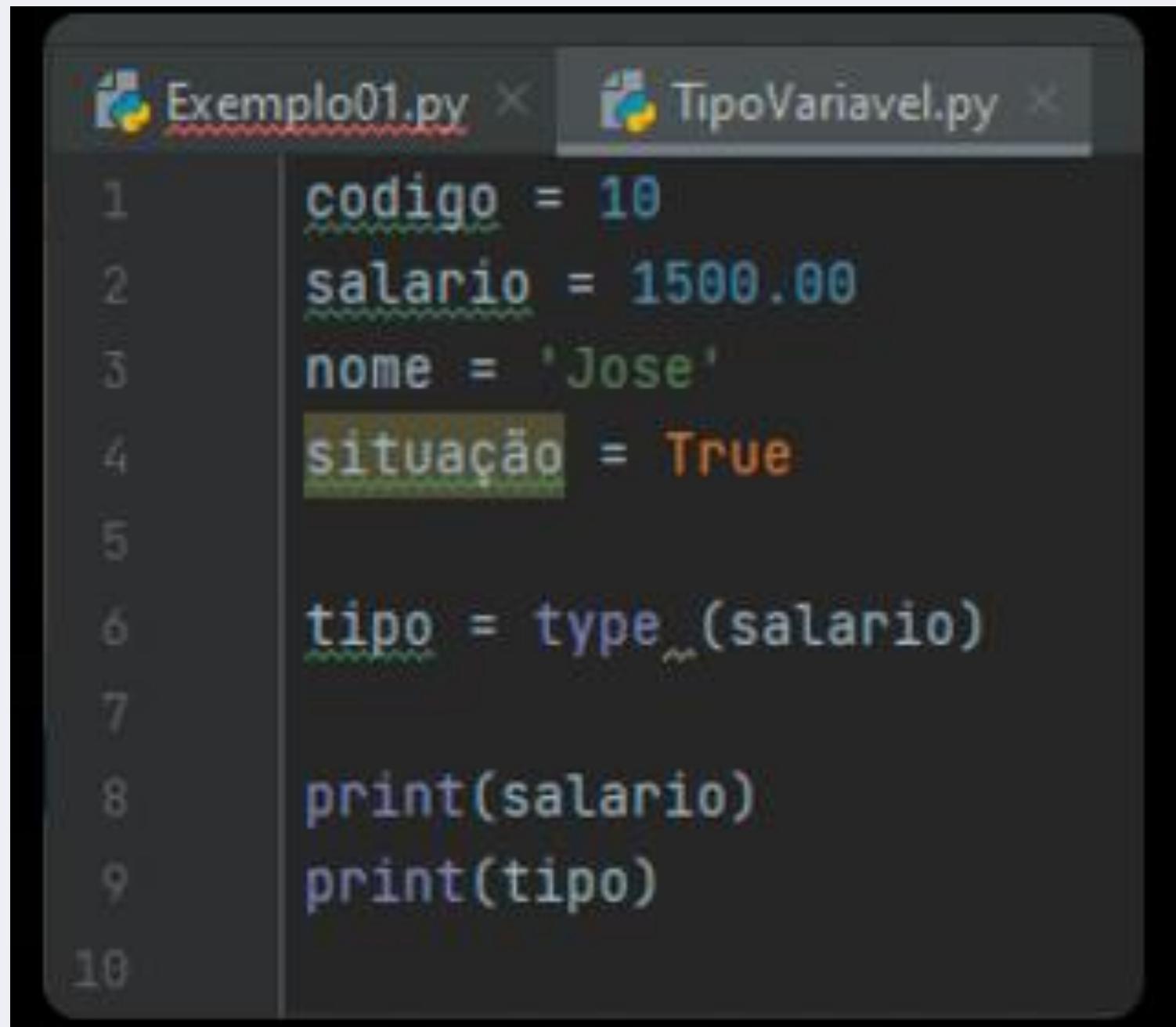
Variáveis em Python não têm tipo fixo, apenas o tipo do conteúdo. Exemplos: X = 10 (inteiro), Cidade = 'Santos' (string).

| Tipo | Exemplo |
|-----------------|--------------|
| Inteiro (int) | X = 10 |
| Real (float) | Y = 3.14 |
| String (str) | Nome = "Ana" |
| Booleano (bool) | Ativo = True |

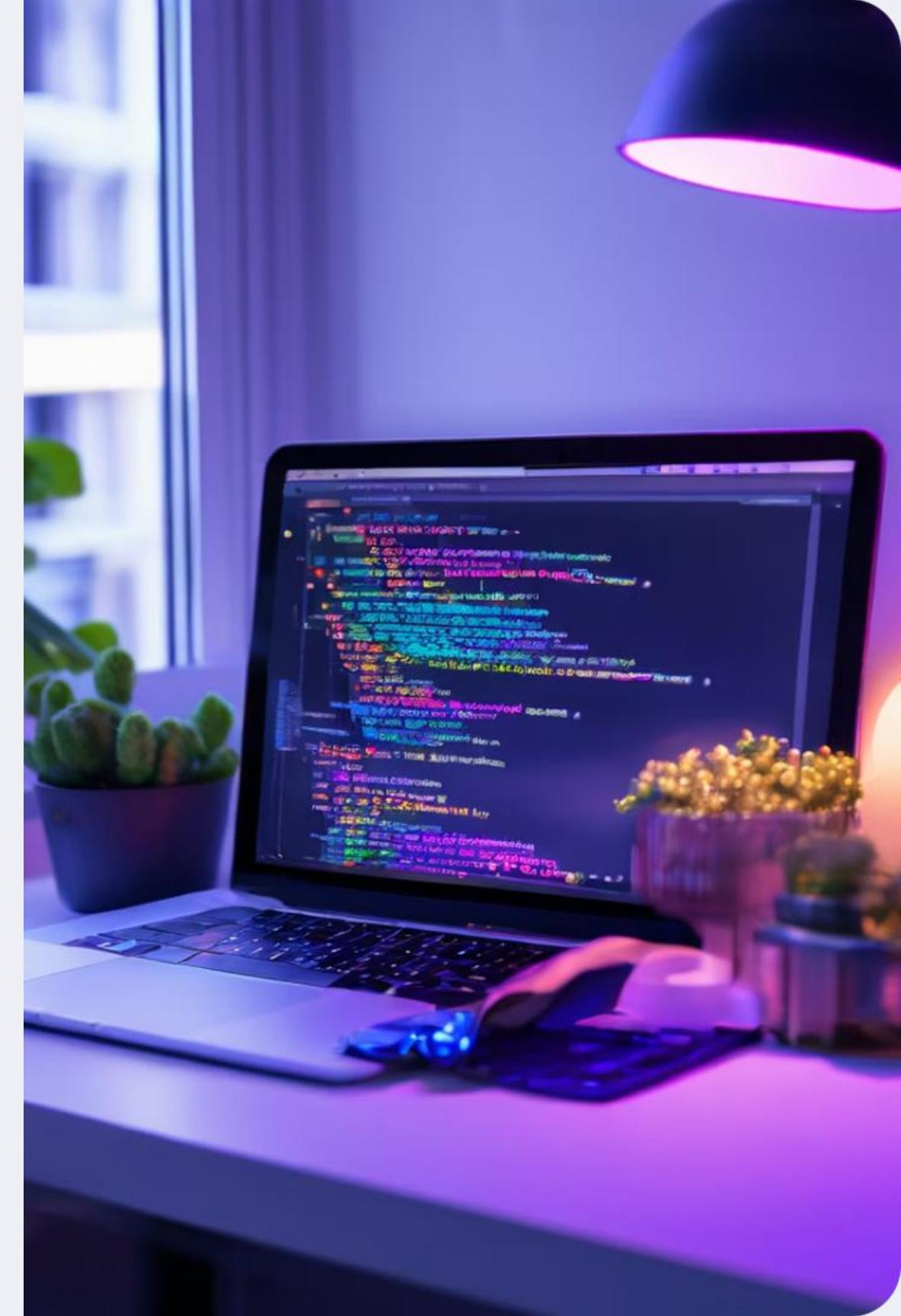


Função type()

A função type() identifica o tipo do valor armazenado em uma variável.

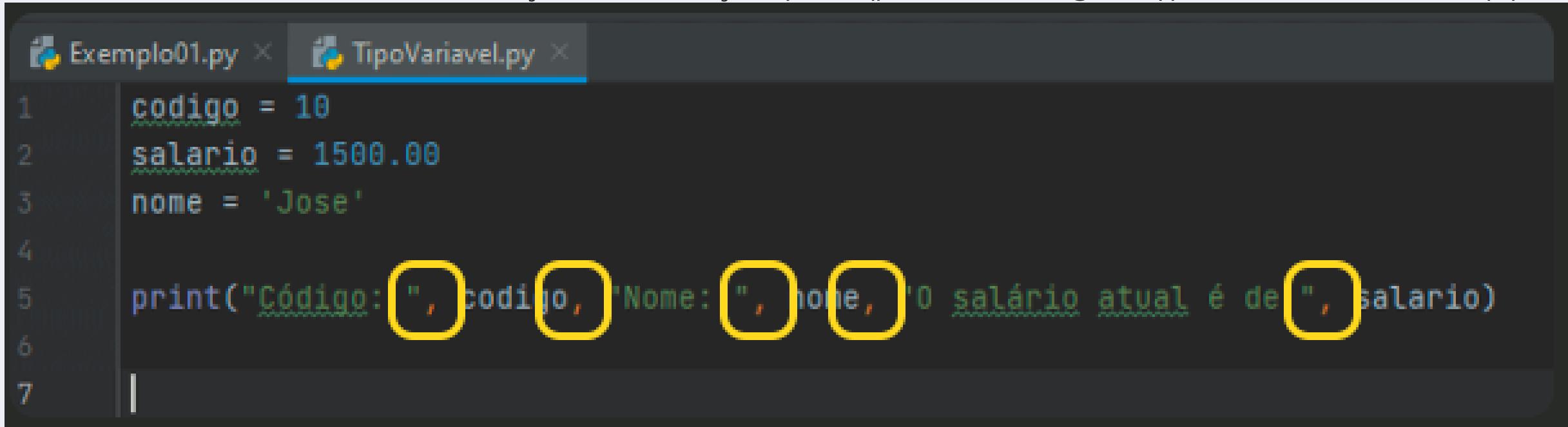


```
Exemplo01.py × TipoVariavel.py ×
1  codigo = 10
2  salario = 1500.00
3  nome = 'Jose'
4  situação = True
5
6  tipo = type(salario)
7
8  print(salario)
9  print(tipo)
10
```

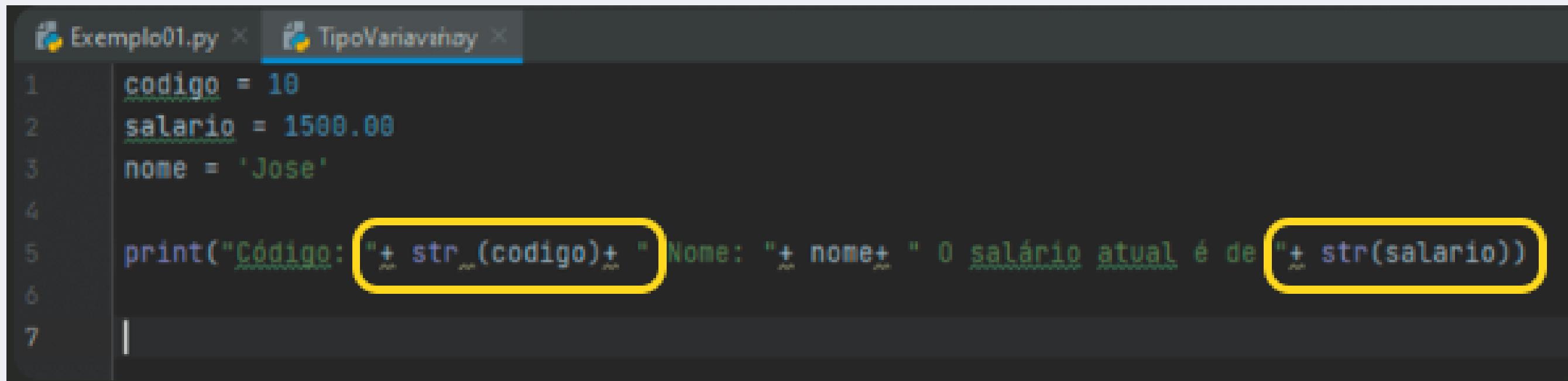


Concatenação de Dados

Podemos concatenar informações na função print() usando vírgula (,) ou sinal de soma (+).



```
Exemplo01.py x TipoVariavel.py x
1     codigo = 10
2     salario = 1500.00
3     nome = 'Jose'
4
5     print("Código: ", codigo, "Nome: ", nome, "O salário atual é de ", salario)
6
7
```



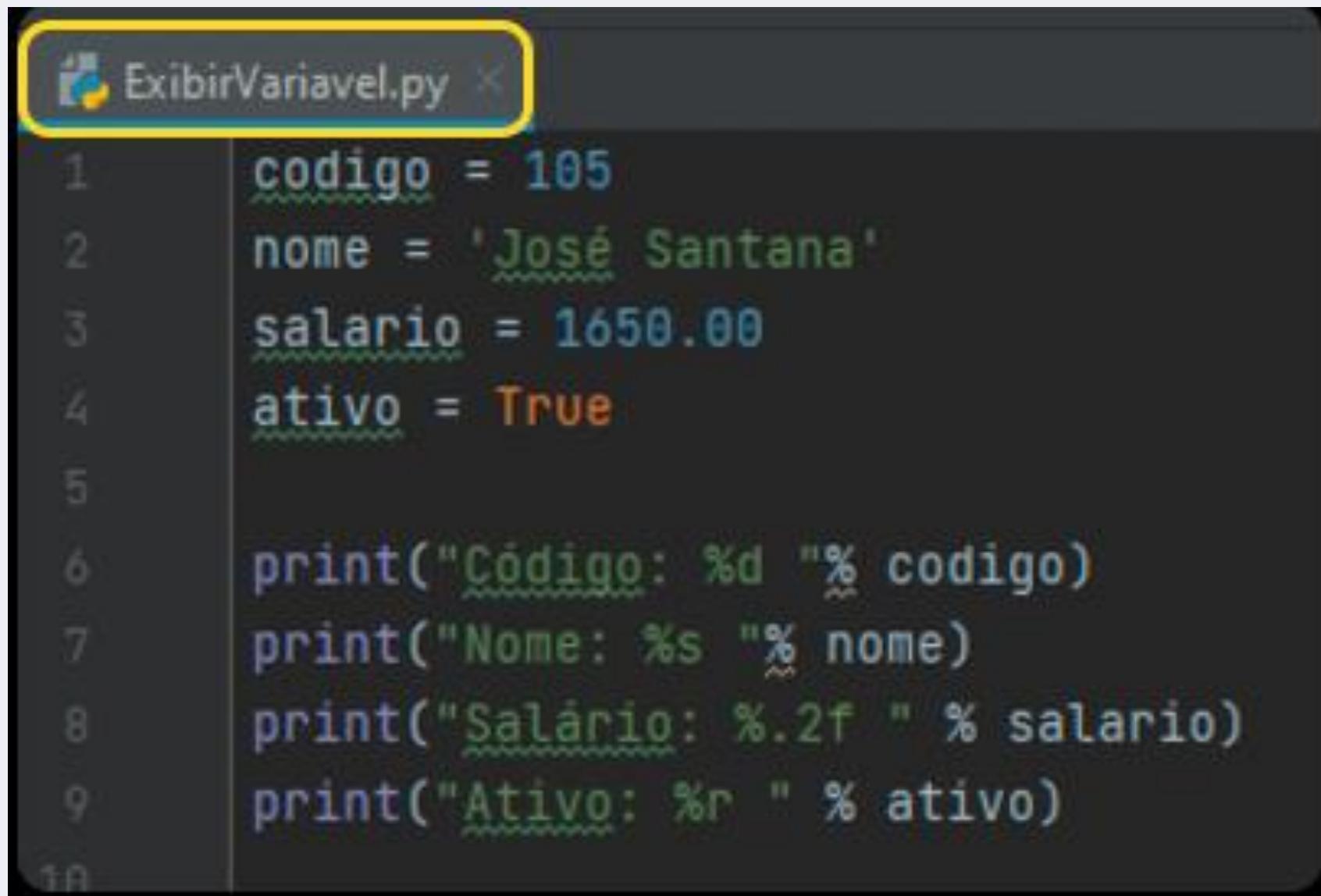
```
Exemplo01.py x TipoVariavel.py x
1     codigo = 10
2     salario = 1500.00
3     nome = 'Jose'
4
5     print("Código: "+ str(codigo)+ " Nome: "+ nome+ " O salário atual é de "+ str(salario))
6
7
```

Máscara de Formatação

| Máscara | Tipo de Dado | Descrição |
|----------|------------------|--|
| %d ou %i | Int (inteiro) | Exibe um valor inteiro. |
| %f | Float ou double | Exibe um valor decimal. |
| %ld | Long Int | Exibe um número inteiro longo. |
| %e ou %E | Float e double | Exibe um número exponencial (número científico). |
| %c | Char (caractere) | Exibe um caractere. |
| %o | Int | Exibe um número inteiro em formato octal. |
| %x ou %X | Int | Exibe um número inteiro no formato hexadecimal. |
| %s | Char | Exibe uma cadeia de caracteres (string). |
| %r | Boolean | Exibe true ou false (verdadeiro ou falso). |

Exemplo de Formatação

Criação de classe ExibirVariavel para demonstrar o uso de máscaras de formatação.

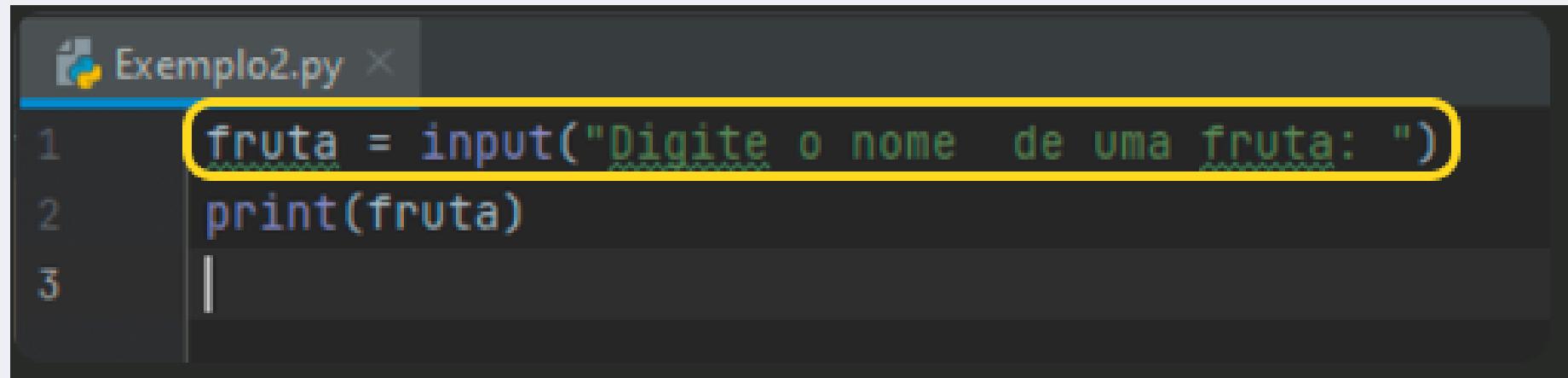


```
ExibirVariavel.py
1  codigo = 105
2  nome = 'José Santana'
3  salario = 1650.00
4  ativo = True
5
6  print("Código: %d" % codigo)
7  print("Nome: %s" % nome)
8  print("Salário: %.2f" % salario)
9  print("Ativo: %r" % ativo)
```

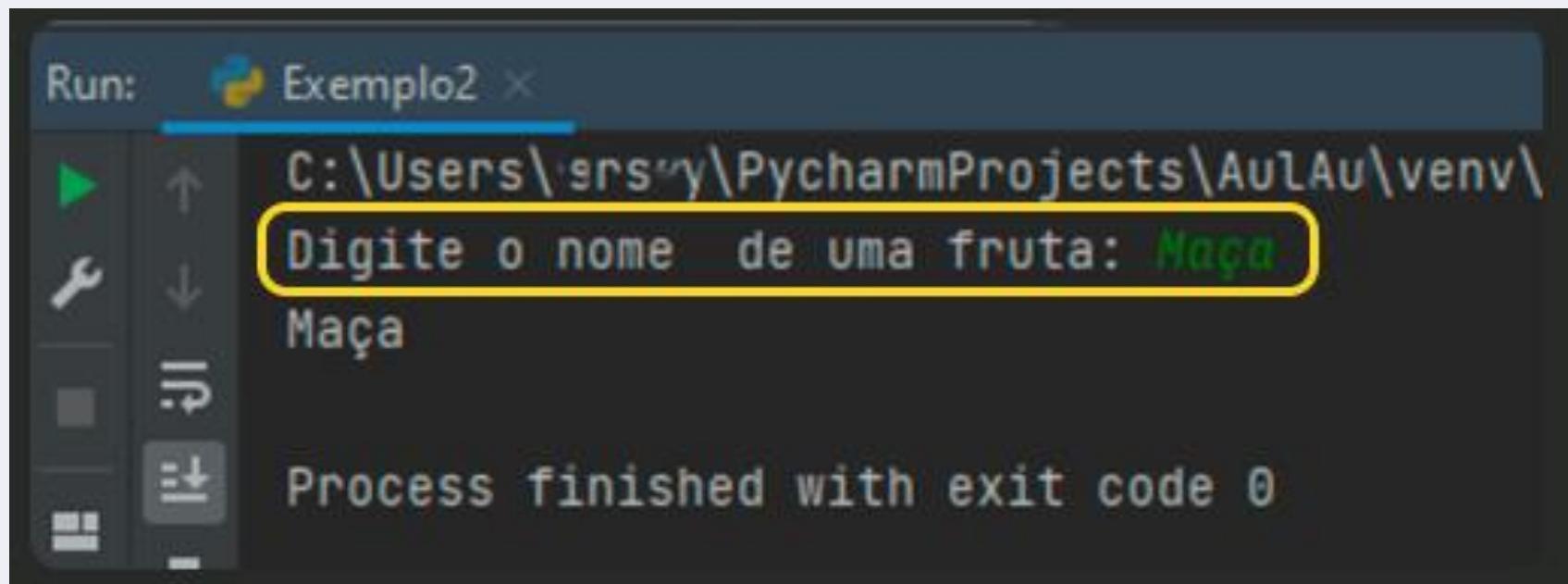


Entrada de Dados

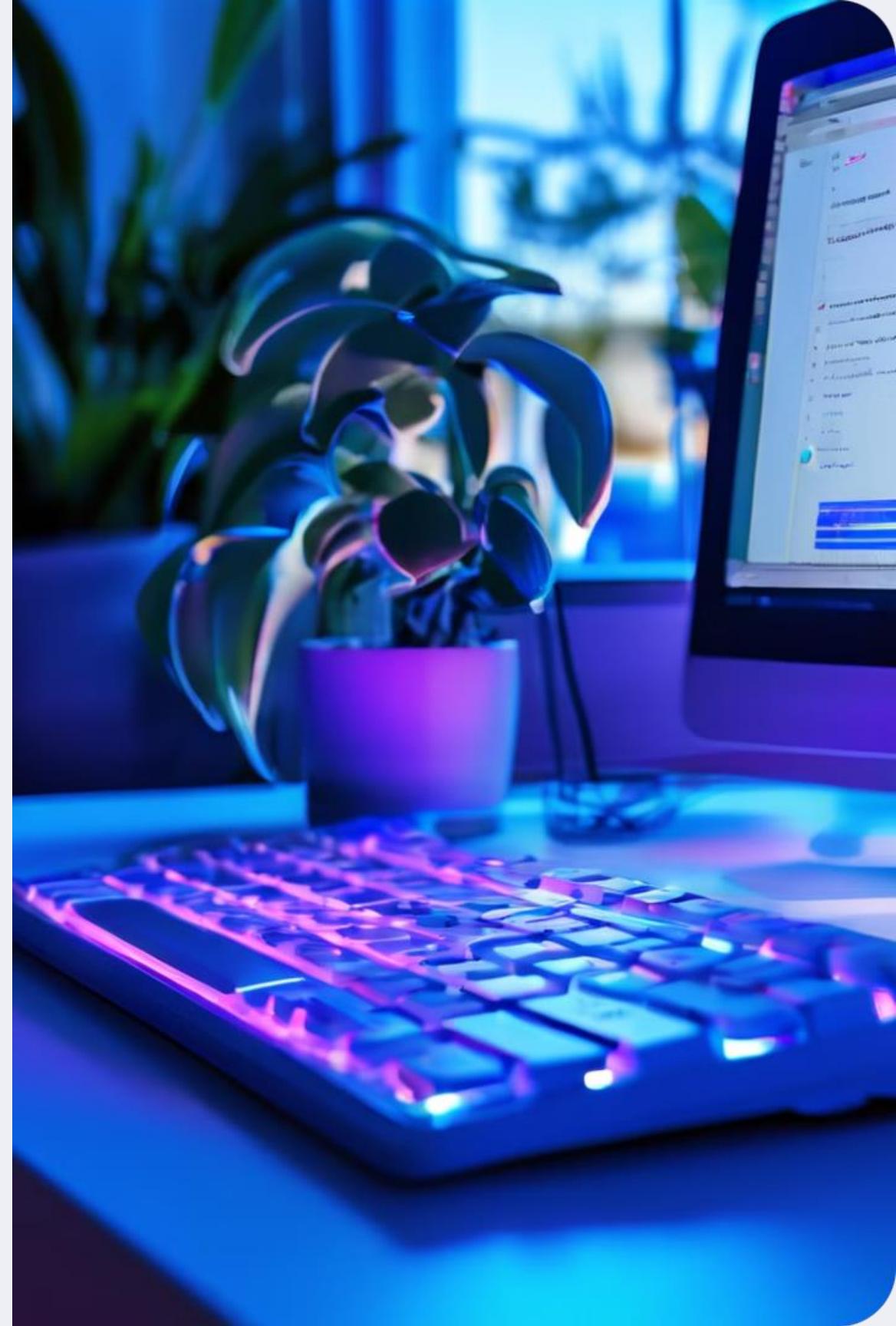
A função `input()` permite interação do usuário, tornando o programa dinâmico.



```
Exemplo2.py
1 fruta = input("Digite o nome de uma fruta: ")
2 print(fruta)
3
```

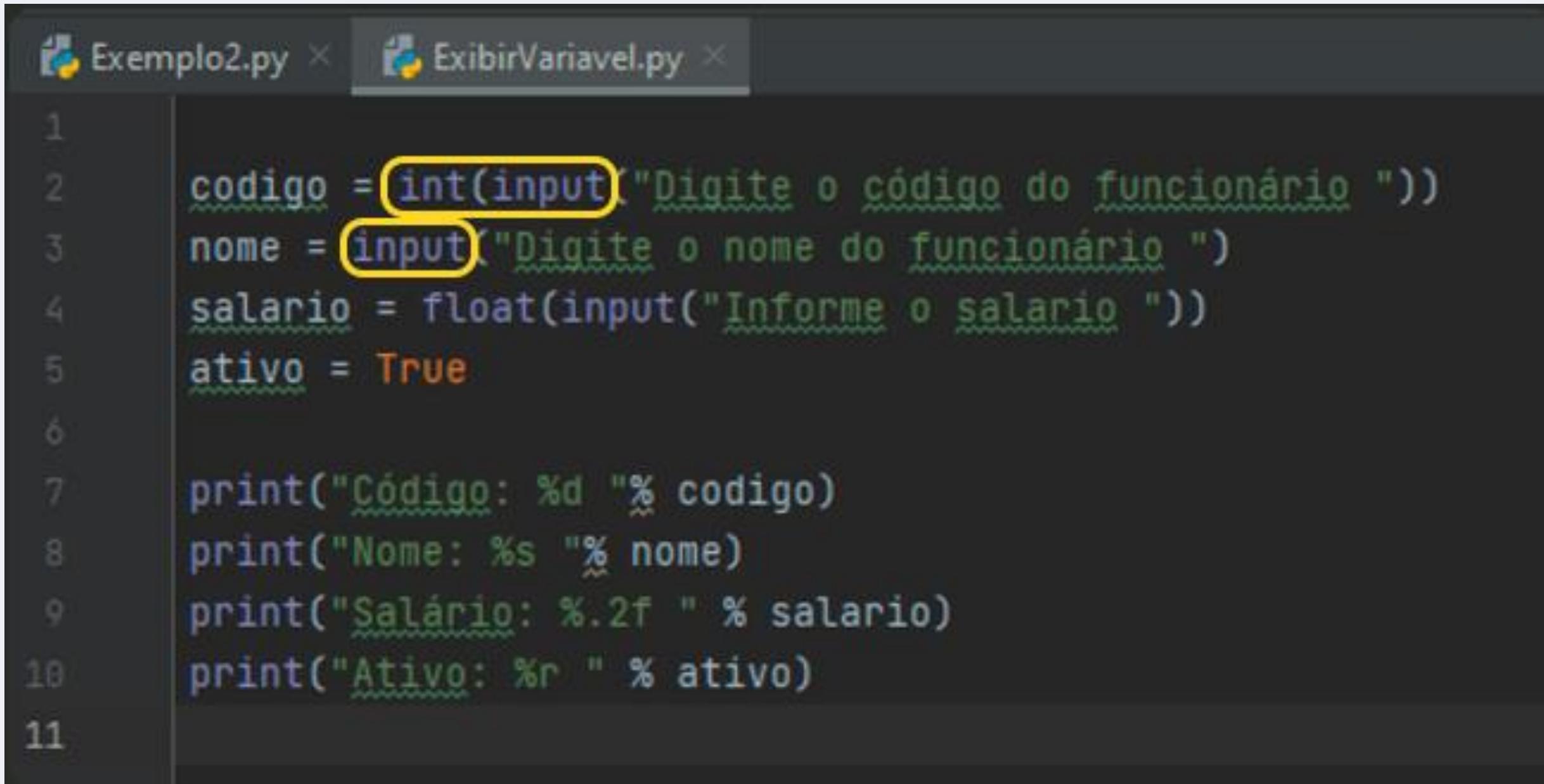


```
Run: Exemplo2
C:\Users\ers\PycharmProjects\AulAu\venv\python Exemplo2.py
Digite o nome de uma fruta: Maça
Maça
Process finished with exit code 0
```



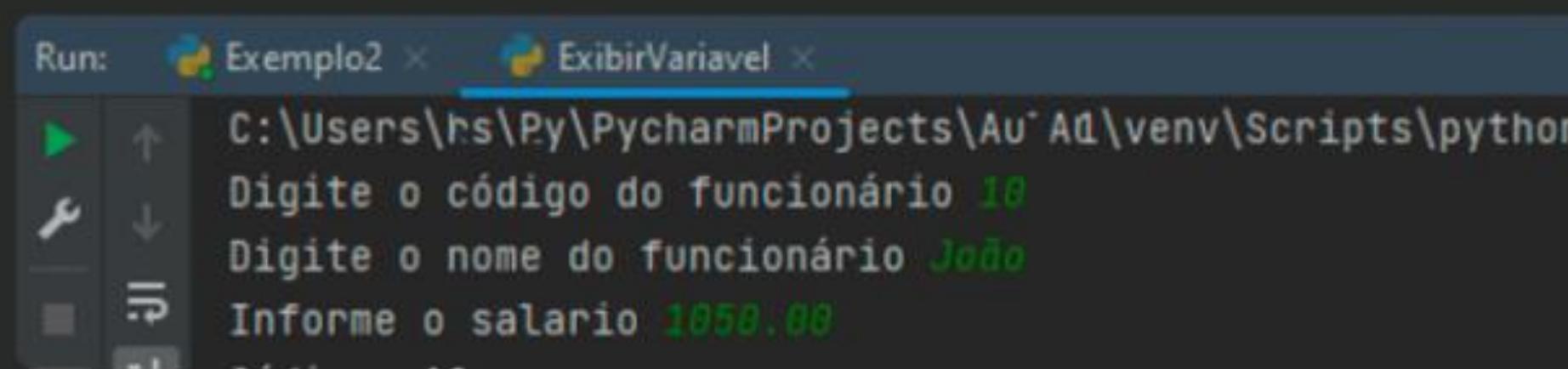
Entrada de Dados Não String

Quando utilizamos a entrada de dados por meio da função `input`, essa será considerada do tipo string. Assim, caso seja necessário realizar a entrada de valores numéricos, temos de converter o tipo de dado, de acordo com o tipo que desejamos armazenar na variável.



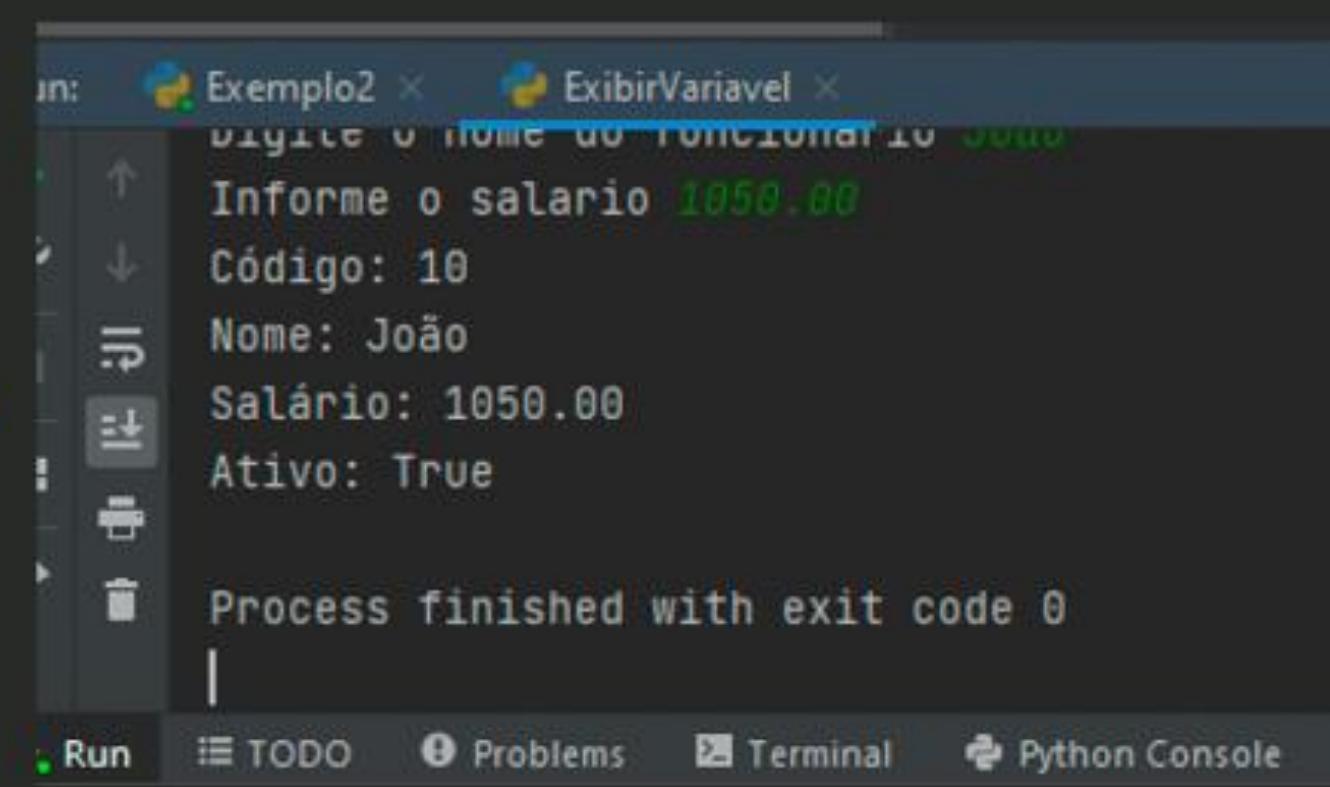
```
1
2     codigo = int(input("Digite o código do funcionário "))
3     nome = input("Digite o nome do funcionário ")
4     salario = float(input("Informe o salario "))
5     ativo = True
6
7     print("Código: %d "% codigo)
8     print("Nome: %s "% nome)
9     print("Salário: %.2f "% salario)
10    print("Ativo: %r "% ativo)
11
```

Ao executar o arquivo é solicitada a entrada das informações por meio do **Console**.



```
Run: Exemplo2 ExibirVariavel
C:\Users\rs\Py\PycharmProjects\Au^ Ad\venv\Scripts\python
Digite o código do funcionário 10
Digite o nome do funcionário João
Informe o salario 1050.00
```

Ao pressionar Enter as informações são exibidas.



```
Run: Exemplo2 ExibirVariavel
Digite o nome do funcionário João
Informe o salario 1050.00
Código: 10
Nome: João
Salário: 1050.00
Ativo: True
Process finished with exit code 0
```

```
1 A = 5
2 B = 15
3 C = 20
4
5 print("A == B AND B > C : ", A == B and B > C)
6 print("A < B OR B > C : ", A < B or B > C)
7 print("not A == B : ", not A == B)
8
```

Run:  Operadores

C:\Users\Py\Py\PycharmProjects\A6tnv\venv\Scripts\python.exe C:/Users/Py/Py/PycharmProjects/A6tnv/Operadores.py

A == B AND B > C : False
A < B OR B > C : True
not A == B : True

Process finished with exit code 0



INDENTAÇÃO



Indentação é uma forma de arrumar o código, fazendo com que algumas linhas fiquem mais à direita que outras, à medida que adicionamos espaços em seu início.



A indentação é uma característica importante no Python, pois além de promover a legibilidade, é essencial para o bom funcionamento do código.



Enquanto na maioria das linguagens, como C, Java e PHP, os blocos de código são delimitados por chaves ({}) ou comandos, em Python, os blocos são delimitados por espaços ou tabulações, formando uma indentação visual.



Não existem símbolos de “abre” e “fecha”.

Exemplo de Indentação

Note que há algumas linhas que ficam mais à direita que outras, à medida que adicionamos espaços em seu início. Isto é o que chamamos de Indentação.

```
idade = int(input("Digite a idade da pessoa: "))

if idade >= 18:
    ➔ print("maior idade")
else:
    ➔ print("menor idade")
```

Note que não há espaços no início de cada linha. Neste exemplo não há Indentação.

```
idade = int(input("Digite a idade da pessoa: "))

if idade >= 18:
    print("maior idade")
else:
    print("menor idade")
```

Caso esteja utilizando uma IDE específica para Python, a indentação é feita, de forma automática, ao pressionar a tecla **Enter**. Por padrão, são usados quatro espaços em branco para definir a indentação.

EstruturaSimples.py

```
1 A = input("Informe um valor para a variável A: ")
2 B = input("Informe um valor para a variável B: ")
3
4 if (A>B):
5     aux=A;
6     A=B;
7     B=aux;
8     print("O valor da variável A agora é : ", A);
9     print("O valor da variável B agora é : ", B);
```

Estrutura de decisão composta

Como já vimos, caso a condição seja **verdadeira**, será executada a **instrução** contida entre o comando **if**.

E se a condição for **falsa**?



Neste caso, utilizaremos o comando **else**, que significa "se não".

Assim, se a condição for falsa, serão executados os comandos que estiverem posicionados logo após a instrução **else**.

Veja o exemplo abaixo:

```
idade = int(input("Digite a idade da pessoa: "))
if idade > 18:
    print("Maior Idade")
else:
    print("Menor Idade")
```

Media.py

```
1 notaA=float(input("Informe a primeira nota: "))
2 notaB=float(input("Informe a segunda nota: "))
3
4 #calcular media
5 mediafinal = (notaA + notaB) / 2
6
7 #verificação
8 if mediafinal >=7.0:
9     print("A Média: %.1f - Aprovado "% mediafinal)
10 else:
11     print("A Média: %.1f - Reprovado "% mediafinal)
```

Utilizando os comandos if - elif - else

Adicionalmente, se existir mais de uma condição alternativa que precise ser verificada, utilizamos a condição **elif**, pois ela avalia as expressões intermediárias antes do comando **else**.

Veja o exemplo:

```
idade = int(input("Digite a idade da pessoa: "))
if idade > 18:
    print("Maior Idade")
elif idade > 16:
    print("Infanto juvenil")
else:
    print("Menor Idade")
```

Estruturas de Repetição

As estruturas de repetição, também conhecidas como laços de repetição, permitem que um conjunto de instruções seja executado, até que uma determinada condição seja verdadeira.

As estruturas de repetição em Python são:

FOR
(para)

WHILE
(enquanto)

Estrutura For (para)

Normalmente utilizamos a estrutura **For** quando sabemos quantas vezes o laço de programação deverá ser executado.

É a estrutura mais utilizada na linguagem Python. Ela aceita sequências estáticas e geradas por **Iteradores**.

Iteradores são estruturas que permitem acesso a vários itens de uma coleção de elementos, de forma sequencial.

Durante a execução de uma estrutura **For**, o contador aponta para um elemento de uma determinada sequência. A cada iteração, o contador é atualizado para que o laço **For** processe os elementos correspondentes.

Sintaxe da estrutura For:

For {referência}

in {sequência}: {bloco de código}

Crescente.py ×

```
1 for n in range(10):  
2     print(n)
```

Run:  Crescente ×

C:\Users\hs\Py
0
1
2
3
4
5
6
7
8
9

No exemplo que acabamos de ver, a variável `n` é inicialmente ajustada para 0 (inicialização com valor padrão).

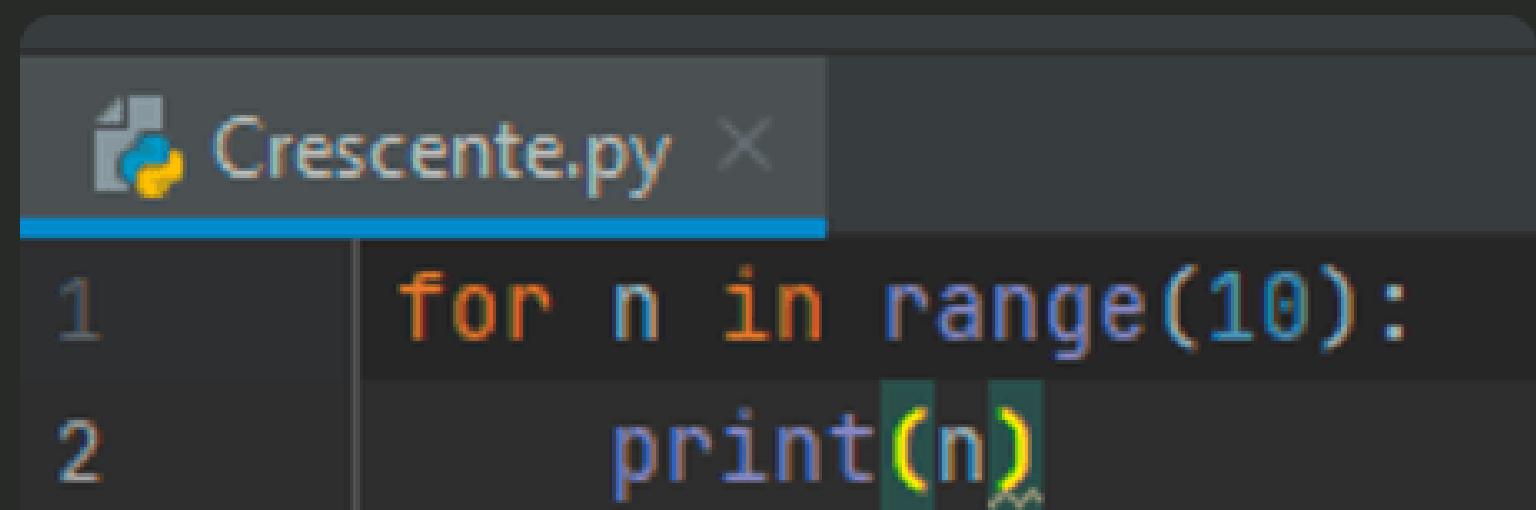
Uma vez que `n` é menor do que 10 (condição), o comando `print` é executado.

Essa condição é adicionada com o comando `range`.

A variável `n` é incrementada em 1 (incremento padrão) e é testado se o valor de `n` ainda é menor do que 10.

O processo se repete até que o valor de `n` fique maior ou igual a 10.

Neste instante, o laço termina e o programa segue a execução das instruções do bloco de repetição.



```
Crescente.py
1 for n in range(10):
2     print(n)
```

Determinar valor inicial

Por padrão, o valor **inicial** do laço de repetição é **0**.

Podemos alterar esse valor no comando **range**.

```
for n in range(5,  
                print(n)
```

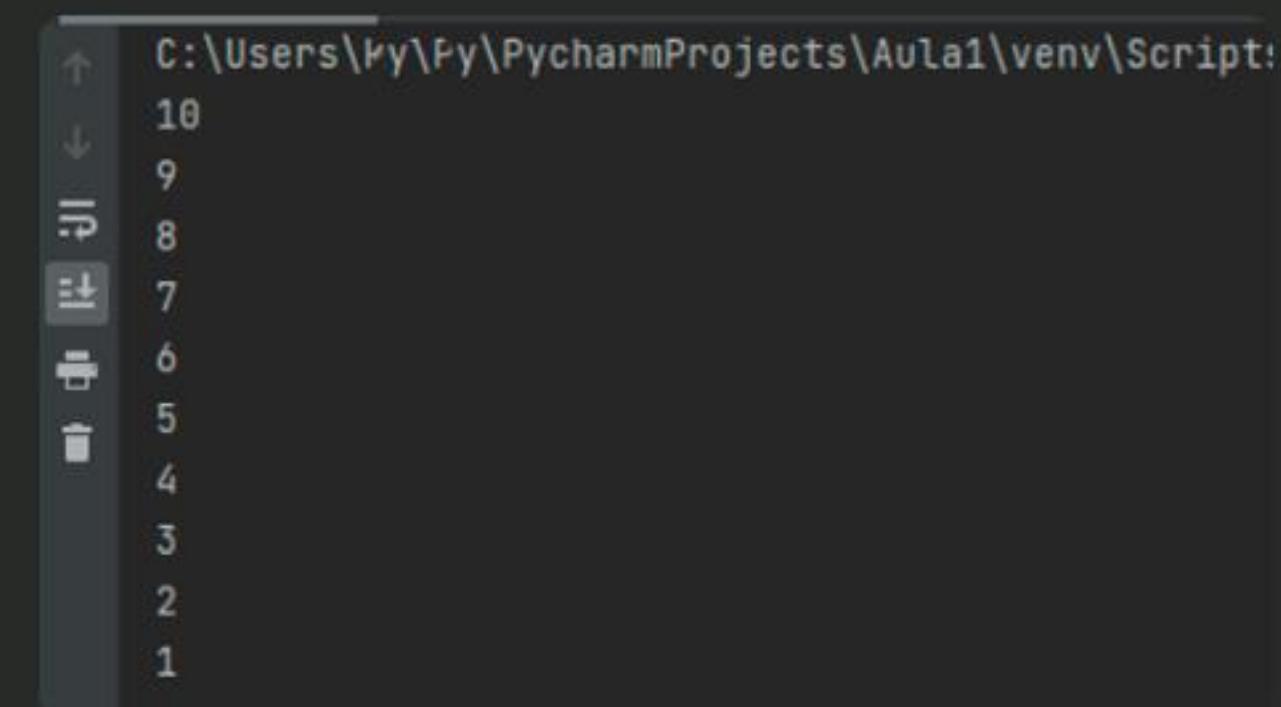
Neste caso, os valores apresentados na tela terão como mínimo, o número **5** e, no máximo, **15**.

Utilizar estrutura em ordem decrescente

Também é possível utilizar o decremento no contador, dentro do comando `range`.

```
for n in range(10, 0, -1):  
    print(n)
```

Neste caso, os valores apresentados na tela estarão em **ordem decrescente**.



```
C:\Users\Py\Py\PycharmProjects\Aula1\venv\Script:  
10  
9  
8  
7  
6  
5  
4  
3  
2  
1
```

Estrutura While (enquanto)

A estrutura **While (enquanto)**, executa um determinado conjunto de instruções, enquanto a condição verificada no início permanecer verdadeira.

Diferente da estrutura **For (para)**, não é necessário determinar o número de vezes que a condição será executada.

No momento em que a condição for falsa, o processamento da rotina é desviado para fora do laço de repetição.

Caso a condição seja falsa, logo no início do laço de repetição, as instruções contidas nele são ignoradas.

Sintaxe da estrutura While:

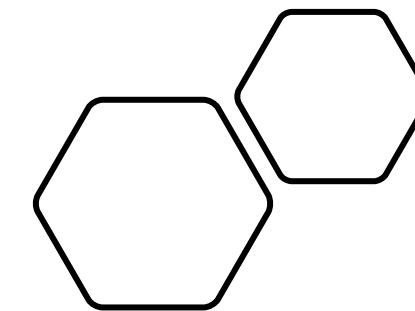
While condição

{bloco de código}



```
1 x = 1
2 while x<=15:
3     print(x)
4     x=x+1
```

```
C:\Users\kn\Py\PycharmProjects\Aula1\venv
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
```



```
MediaValores.py
1 qtd = 0
2 soma = 0
3 media = 0
4 valor = float(input("Digite um valor: "))
5
6 while (valor > 0.0):
7     soma = soma + valor
8     qtd = qtd + 1
9     # Entrada de valores
10    valor = float(input("Digite um valor: "))
11
12 #caso digite um valor negativo o laço encerra
13 media = soma / qtd
14 print("\n Total da Soma: ", soma)
15 print("\n Quantidade de valores digitados: ", qtd)
16 print("\n Média dos valores: ", media)
```

```
MediaValores
C:\Users\k\Py\PycharmProjects\Aula1\ven
Digite um valor: 25.6
Digite um valor: 52.5
Digite um valor: 55.9
Digite um valor: -1
Total da Soma: 112.0
Quantidade de valores digitados: 3
Média dos valores 37.3:
```

Indentação

If - elif - else

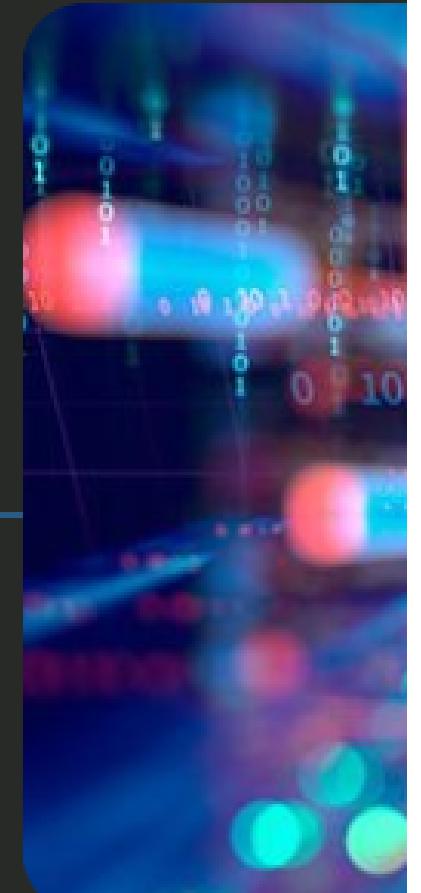
For
(para)

While
(enquanto)

Funções

Funções são estruturas que possibilitam a separação da programação em blocos.

A implementação de funções em programas é fundamental, pois tem o objetivo de otimizar o código-fonte e evitar a replicação do código, ou seja, definimos as funções uma única vez e sempre que necessário podemos utilizá-las.



Sintaxe para aplicação de uma função

A declaração de uma função é dividida em três partes: nome, parâmetros e corpo.

```
def nome_da_função (parâmetros):  
    <instruções>  
    return "valor do retorno"
```

```
def nome_da_função (parâmetros):  
    <instruções>  
    return "valor do retorno"
```

>>1. def

A palavra **def** determina o início da função.

>>2. Parâmetros

São informações que a função pode receber para o seu processamento.

Os parâmetros podem existir ou não.

>>3. Corpo da função

É o local em que é aplicada a sequência de instruções, como entrada, processamento e/ou saída.

>>4. return

Deve ser utilizado quando houver necessidade de retornar alguma informação para a ação da função.

ExemploFunção.py > ...

```
1  def mensagem1():
2      print("Hello World")
3
4  def mensagem2():
5      return 'Olá Mundo'
6
7  mensagem1()
8
9  texto=mensagem2()
10 print(texto)
11
```

Para melhor compreensão sobre funções na linguagem Python, vamos criar um programa para cálculo de média de notas.

No programa, o usuário deverá digitar duas notas para o aluno. Na sequência, o programa irá calcular e exibir a média, informando se o aluno está aprovado (média maior ou igual a 7.0) ou reprovado.

O programa deverá ter uma função para leitura das notas e outra, para cálculo da média.

```
Media.py
1 def lernotas():
2     n=float(input('Digite uma nota para o aluno(a): '))
3     return n
4
5
6 def resultado(n1,n2):
7     media=(n1+n2)/2
8     print("Nota 1: ", n1)
9     print("Nota 2: ", n2)
10    print("Média: ", media, "Resultado: ", end="")
11    if media >= 7:
12        print("Aprovado")
13    else:
14        print("Reprovado")
15
16
17 a = lernotas()
18 b = lernotas()
19 resultado(a,b)
```