

Unidade II

3 ESTRUTURA CONDICIONAL

Na unidade I, aprendemos sobre a estrutura sequencial, que funciona como uma "receita de bolo", uma instrução executada após a outra. Neste tópico, estudaremos a estrutura condicional ou desvio condicional, como também é conhecida. Essa estrutura é usada para decidir se um conjunto de instruções deve ou não ser executado.

3.1 If, elif e else

Uma estrutura de decisão testa uma expressão lógica e, caso ela seja verdadeira, será executado um conjunto de instruções definido. A decisão (ou o caminho) será executada em função do valor (lógico) que a expressão pode assumir. Os tipos de instruções utilizadas na linguagem Python são as instruções if, if-else e elif.

Trata-se da instrução mais versátil e conhecida em estruturas de decisão independentemente da linguagem de programação. A instrução if causa um desvio do fluxo de execução das instruções de um programa, em função do resultado de uma condição. Essa condição nada mais é que uma expressão lógica. Sua sintaxe, na forma mais simples, é:

```
<instrução 1>  
if <condição>:  
    <instrução 2>  
    <instrução 3>  
<instrução 4>
```

É importante notar que, em Python, os blocos de instruções são separados por indentação. Indentar é o recuo do texto em relação a sua margem, ou seja, antes de escrevermos uma instrução, utilizamos quatro espaços da margem esquerda até a instrução. No caso apresentado, as instruções 2 e 3 somente serão executadas se a <condição> for verdadeira. Caso a condição seja falsa, somente as instruções 1 e 4 serão executadas.

Como dito, se a condição for avaliada como verdadeira, será executado o bloco de instruções indentadas (2 e 3). Após a execução da instrução if, o fluxo é transferido para a próxima instrução.

As estruturas de decisão utilizadas em Python são:

Estrutura de decisão simples:

```
if <condição>:  
    <bloco_execução>
```

Estrutura de decisão composta:

```
if <condição>:  
    <bloco_execução 1>  
else:  
    <bloco_execução 2>
```

O uso da palavra-chave "else" na estrutura composta é opcional e seu conteúdo será executado somente se o resultado da expressão lógica booleana for igual a false.

Estrutura de decisão aninhada:

```
if <condição 1>:  
    <bloco_execução 1>  
elif <condição 2>:  
    <bloco_execução 2>  
else:  
    <bloco_execução 3>
```



Observação

Cada bloco de execução é uma sequência de comandos, um após o outro. Todos os comandos do bloco são indentados. Sem a indentação, a estrutura condicional não funciona.

Exemplo 1, decisão simples:

```
a = 33  
b = 200  
if a < b:  
    print("a é menor que b")
```

Note que, no exemplo 1, a condição é verdadeira e a instrução print será executada. O comando print() tem indentação de quatro espaços.

Exemplo 2, decisão composta:

```
numero = 10
if numero < 10:
    print("O valor de número é menor que: " + str(numero))
else:
    print("O valor de número é maior ou igual a: " + str(numero))
```

No exemplo 2, a variável `numero` guarda o valor 10 e a condição `numero < 10` é falsa; sendo assim, o bloco de código do `else` é executado.

Exemplo 3, decisão aninhada:

```
a = 5
if a < 5:
    print("O valor de a é menor que: " + str(a))
elif a > 5:
    print("O valor de a é maior que: " + str(a))
else:
    print("O valor de a é igual a: " + str(a))
```



Observação

No caso da decisão aninhada, é possível colocar vários blocos `elif`. Não existe limite para a quantidade desses blocos.

No caso do exemplo de decisão aninhada, temos dois testes de condição, um no `if` e outro no `elif`, e caso as duas forem falsas o bloco de código do `else` é executado.

Existe uma forma curta de codificar a decisão simplificada, em uma única linha, porém ela é desencorajada caso a instrução seja muito grande, porque pode atrapalhar a legibilidade do código.

Exemplo:

```
if a == b: print("a é igual a b")
```

Também existe a forma curta de codificação da decisão composta. Ela é diferente de todas as outras formas aqui já apresentadas. O comando executado, quando a condição do `if` é verdadeira, vem antes do `if`. Já o comando executado, quando a condição é falsa, vai depois do `else`. Lembrando que os: também não são usados.

Exemplo:

```
a = 45  
b = 134  
print("a é maior") if a > b else print("b é maior")
```



Observação

A técnica apresentada na decisão composta curta em uma linha é conhecida como operadores ternários ou expressão condicional.

A utilização dos operadores lógicos AND e OR será demonstrada nos próximos dois exemplos. É possível fazer qualquer combinação e quantidade deles em uma expressão lógica.

Exemplo 1:

```
a = 86  
b = 26  
c = 129  
if a > b and c > a:  
    print("As duas condições são verdadeiras.")
```

Exemplo 2:

```
a = 86  
b = 26  
c = 129  
if a > b or c > a:  
    print("Uma ou as duas condições são verdadeiras.")
```

O último exemplo detalha a utilização de decisões encadeadas, ou seja, dentro de um bloco de comandos if tem outro if. Pode existir qualquer quantidade de encadeamento, desde que cada nível adicione a quantidade de espaços da indentação. No primeiro nível, temos 4 espaços; no segundo, 8 espaços; no terceiro, 12 e assim por diante.

Exemplo:

```
x = 15  
  
if x > 5:  
    print("Maior que 5,")  
if x > 10:  
    print("e também maior que 10.")  
else:  
    print("e menor que 10.")
```



Lembrete

Não esqueça a indentação para que if funcione corretamente. A linguagem não nos obriga a usar quatro espaços, mas tem que manter a quantidade escolhida até o fim. A recomendação é quatro ou dois espaços, ou uma tabulação.

3.2 Pass

O Python exige que a estrutura condicional if tenha pelo menos uma instrução. Às vezes é conveniente criar uma estrutura condicional vazia, ou dentro do if, elif ou else. O pass é uma instrução vazia, ou seja, não executa nada. Utilizar o pass dentro do bloco if evita que o Python reclame com um erro.

Exemplo 1:

```
a = 10
b = 20
if b > a:
    # para programar mais tarde
else:
    print("a é maior que b")
```

O exemplo 1 não funciona, pois o comentário dentro do if não é considerado um comando, o Python simplesmente ignora os comentários. Veja um caso de uso do pass no exemplo 2.

Exemplo 2:

```
a = 10
b = 20
if b > a:
    # para programar mais tarde
    pass
else:
    print("a é maior que b")
```

3.3 Switch–case

O switch-case é uma estrutura de decisão que não é essencial para a programação, pois a estrutura if-elif-else é suficiente. Por isso, o Python não tem uma estrutura switch-case.

A estrutura switch-case é utilizada em várias linguagens de programação. Ela define o código a ser executado, levando em conta uma comparação de valores.

Segue a sintaxe da estrutura em linguagem C#:

```
switch (<variável>)  
{  
    case <valor1>:  
        // bloco de código 1  
        break;  
    case <valor2>:  
        // bloco de código 2  
        break;  
    default:  
        // bloco de código 3  
        break;  
}
```

Na primeira linha, temos a variável cujo valor vamos comparar nos cases, diferentemente do if-elif-else, que utiliza uma expressão. O valor da variável é comparado com os valores dos cases e caso seja igual a algum deles, o bloco é executado, só parando a execução do bloco no comando break. Caso o valor não seja igual a nenhum dos cases, é executado o bloco após a instrução default. O bloco default é opcional na estrutura switch-case.

Está sendo considerada a inclusão de uma estrutura match-case no Python 3.10, prevista para outubro de 2021. O principal motivo é facilitar a manipulação de dados estruturados em JSON, dicionários, objetos etc., usando padrões e expressões regulares. Essa estrutura poderá ser usada exatamente como a switch-case.

Exemplo de aplicação

Apesar de o Python não ter uma estrutura switch-case, é possível criar uma utilizando outros recursos mais avançados de Python. Fica a dica para um exercício!

4 ESTRUTURA DE REPETIÇÃO

As estruturas de repetição são ferramentas poderosas na programação. Elas permitem executar um bloco de comandos repetidamente até que uma condição definida pelo programador seja atingida. As principais estruturas de repetição que vamos estudar são while e for. Elas também são conhecidas como estruturas de laço.

4.1 While

A estrutura de repetição while testa uma expressão lógica; se ela for verdadeira, o bloco de instruções do while é executado. Segue a sintaxe do while:

```
while <expressão lógica>:  
    <bloco de comandos>
```

Toda vez que o bloco de comandos/instruções termina sua execução, a expressão lógica é testada novamente. No momento em que a expressão tem o valor falso, o bloco não é mais executado e o fluxo sequencial do programa continua da próxima instrução, que está fora do bloco de instruções.

Exemplo:

```
i = 1  
while i < 10:  
    print(i)  
    i += 1  
print("Fim de laço while!")
```

Segue a saída do programa exemplo:

```
1  
2  
3  
4  
5  
6  
7  
8  
9  
Fim do laço while!
```



Observação

Lembre-se sempre de incrementar a variável da condição. No caso do exemplo, i é incrementado de 1 sempre que passa por i += 1, caso contrário terá um laço infinito.



Saiba mais

Temos uma comunidade de programação chamada Python Brasil que reúne grupos de usuários em todo o Brasil interessados em difundir e divulgar a linguagem Python.

Para saber mais sobre a Python Brasil, acesse:

Disponível em: <https://cutt.ly/pmSuRgf>. Acesso em: 13 jul. 2021.

4.1.1 Break/continue

As instruções `break` e `continue` podem ser usadas dentro de estruturas de laços como `while` e `for` para, respectivamente, saírem do laço completamente e irem para próxima iteração imediatamente. O comando `break` pode parar o laço mesmo que a condição do `while` seja verdadeira. Vejamos o exemplo.

Exemplo:

```
i = 1
while i < 10:
    print(i)
    if i == 5:
        break
    i += 1
print("Fim do laço while pelo break!")
```

Segue a saída do programa exemplo:

```
1
2
3
4
5
Fim do laço while pelo break!
```

O comando `continue` pode parar a iteração corrente e começar a próxima. Segue o exemplo:

```
i = 1
while i < 10:
    i += 1
    if i == 5:
        continue
    print(i)
print("Fim do laço while!")
```


Segue a saída do programa exemplo. Note que o número 5 não é impresso, já que o continue para a execução do laço quando `i == 5` e vai para a próxima iteração, não permitindo o `print()` executar:

```
2
3
4
6
7
8
9
10
Fim do laço while!
```

4.2 For

O laço `for` é utilizado, na linguagem Python, para interagir com uma estrutura de dados sequencial, que pode ser uma lista, um dicionário, um conjunto, uma tupla ou uma string. Como até agora só conhecemos as strings, os exemplos desta unidade vão interagir com listas de string ou números, a função `range()` e a string.

O `for` funciona de forma diferente em Python comparado a outras linguagens. Ele trabalha mais como um método iterativo, assim como encontrado em outras linguagens orientadas a objeto.

Com a estrutura `for` podemos executar um bloco de instruções para cada item contido numa lista, conjunto, string etc. Segue a sintaxe da estrutura `for`:

```
for <variável> in <objeto iterável>:
    <bloco de instruções>
```

Exemplo de iteração sobre os caracteres de uma string:

```
for letra in "banana":
    print(letra)
```

Saída:

```
b
a
n
a
n
a
```

O laço é executado seis vezes, uma para cada letra da palavra "banana".



Lembrete

A variável de iteração do laço for pode ter qualquer nome válido em Python. Não é mágica ter o nome "letra", em nosso exemplo, e imprimir letra a letra da palavra!

Listas são uma estrutura de dados bem flexível em Python. Elas serão estudadas nas próximas unidades, com mais detalhes. No próximo exemplo, vamos usar uma lista de string. As listas são representadas por colchetes e cada item é separado por vírgulas. Exemplos: ["a", "b", "c"], [1, 2, 3].

Exemplo 1:

```
frutas = ["maça", "banana", "melancia"]
for x in frutas:
    print(x)
```

Saída:

```
maça
banana
melancia
```

Exemplo 2:

```
for a in [10, 2, 100, 32]:
    print(a)
```

Saída:

```
10
2
100
32
```

A função `range()` retorna uma sequência de números, começando em 0, incrementando 1 e terminando no número especificado. O começo em 0 e o incremento em 1 podem ser modificados, dizendo à função que queremos outro começo e/ou outro incremento. Essa função facilita a criação de sequências numéricas, sem o desenvolvedor ter que escrever ou programar um laço para criá-las.

Exemplo 1:

```
for x in range(5):  
    print(x)
```

Saída:

```
0  
1  
2  
3  
4
```

Exemplo 2:

```
# começa em 5 e termina em 10, sem imprimir o 10  
for x in range(5, 10):  
    print(x)
```

Saída:

```
5  
6  
7  
8  
9
```

Exemplo 3:

```
# começa em -10 e termina em -100, e o incremento é -30  
for x in range(-10, -100, -30):  
    print(x)
```

Saída:

```
-10  
-40  
-70
```

Dizemos que um objeto é iterável, isto é, candidato a ser alvo de uma função ou estrutura que espera algo capaz de retornar sucessivamente seus elementos, um de cada vez. Nós vimos que o comando `for` é um exemplo de estrutura, enquanto um exemplo de função que recebe um iterável é a função `sum()`, que soma uma sequência:

Exemplo 1:

```
print(sum([1, 2, 3]))
```

Saída:

```
6
```

Exemplo 2:

```
# soma 0, 1, 2 e 3  
print(sum(range(4)))
```

Saída:
6

Também podemos colocar um for dentro de um bloco de instruções for. São chamados de for encadeados ou aninhados. O mesmo pode ser feito com o laço while.

Exemplo:

```
tamanho = ["pequena", "média", "grande"]  
frutas = ["maça", "banana", "melancia"]  
for x in tamanho:  
    for y in frutas:  
        print(x, y)
```

Saída:
pequena maça
pequena banana
pequena melancia
média maça
média banana
média melancia
grande maça
grande banana
grande melancia



Observação

As instruções pass, break e continue também funcionam no laço for.



Saiba mais

A Olimpíada Brasileira de Informática, para alunos cursando do quarto ano do Ensino Fundamental até o primeiro ano do Ensino Superior, tem uma modalidade de programação que aceita a utilização de Python.

Para saber mais sobre a Olimpíada Brasileira de Informática, acesse:

Disponível em: <https://cutt.ly/mmSuAa9>. Acesso em: 13 jul. 2021.

Na unidade III, estudaremos mais a fundo as strings e as ferramentas para sua manipulação, assim como o conceito de funções e como criar suas próprias. Também veremos como abrir, ler e escrever em arquivos, assim como trabalhar com estruturas de dados mais sofisticadas como conjuntos, tuplas e listas.

4.3 Exercícios sugeridos

1. Crie um programa que solicite um número inteiro ao usuário e imprima na tela uma mensagem dizendo se o número é positivo ou negativo.
2. Construa um programa em Python que solicite uma idade entre 0 e 130 anos. Imprima uma mensagem se o valor for inválido e continue solicitando até que o usuário digite um valor válido.
3. Crie um programa em Python que solicite ao usuário dois números inteiros e imprima o maior deles.
4. Faça um programa que solicite o nome do usuário e a senha, imprimindo uma mensagem de erro e voltando a solicitar uma nova senha caso ela seja igual ao nome do usuário.
5. Crie um programa que verifique se uma letra digitada pelo usuário é uma vogal, consoante ou número.
6. Construa um programa que leia do teclado e valide os seguintes dados sobre funcionários de uma empresa:
 - a) Nome: maior que três caracteres.
 - b) Idade: entre 0 e 130.
 - c) Salário: maior que zero.
 - d) CEP: tamanho 8.
 - e) Departamento: 'Administrativo' ou 'Fabrica'.
7. Crie um programa que verifique se uma letra digitada é "S" para sair do programa. Caso a letra digitada for diferente de S, o programa continua perguntando se o usuário deseja sair.
8. Supondo que José tenha R\$ 75.000,00 investidos a uma taxa anual de 3% de juros e que João tenha R\$ 215.000,00 investidos com uma taxa de 1,5% de juros anuais, faça um programa que calcule e escreva quantos anos seriam necessários para que José ultrapasse ou iguale João, mantidas as taxas de juros.

9. Altere o programa anterior permitindo ao usuário informar o capital inicial e as taxas de juros. Os capitais devem ser positivos e o juros do primeiro devem ser maiores que os do segundo.

10. Crie um programa que solicite três números aos usuários e imprima na tela o menor deles.

11. Construa um programa para que receba duas notas de um aluno. O programa deve processar a média e mostrar a mensagem:

a) "Aprovado", para média 7,0 ou maior.

b) "Reprovado", para média menor que 7,0.

12. Construa um programa que mostre na tela os números ímpares de 1 a 30, um por linha. Depois mostre os mesmos números impressos na mesma linha.

13. Crie um programa que pergunte o preço do arroz cateto, agulhinha e jasmim e mostre na tela qual você deve comprar (sempre será o arroz com o menor preço).

14. Crie um programa que pergunte ao usuário 10 números e imprima na tela o maior deles.

15. Crie um programa que pergunte ao usuário cinco números e imprima na tela o maior e o menor deles.

16. Crie um programa que pergunte ao usuário seis números e imprima na tela a soma e a média deles.

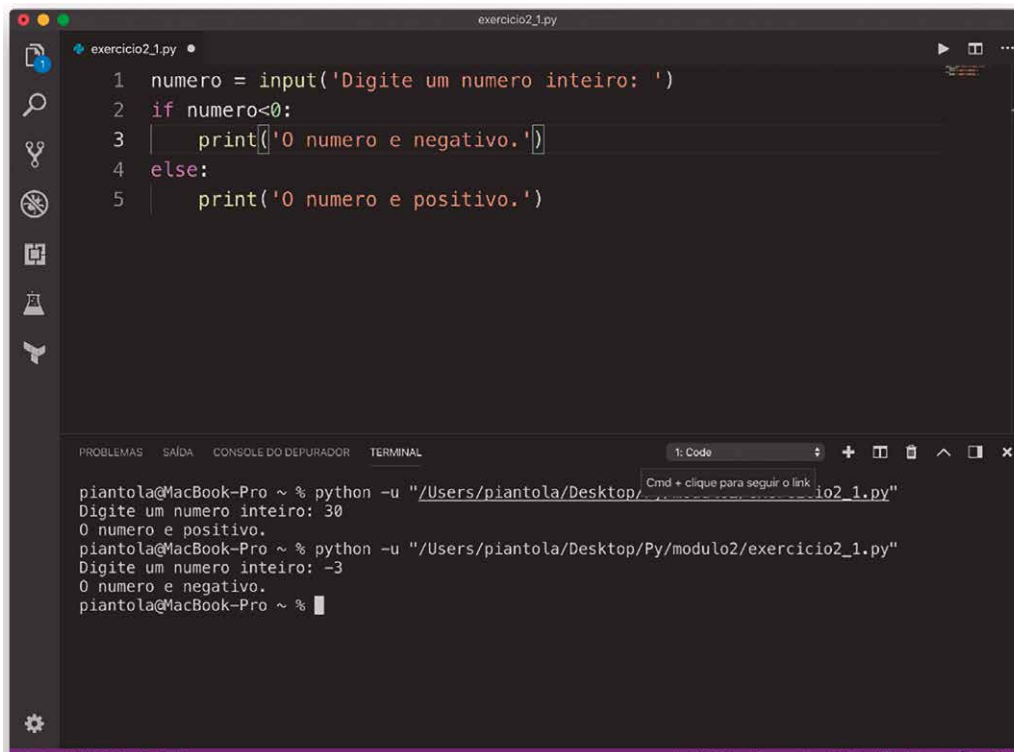
17. Crie um programa que pergunte a hora ao usuário (0-24). Imprima na tela o texto: "Bom dia.", "Boa tarde." ou "Boa noite." ou "Hora Inválida.", de acordo com a resposta.

18. Crie um programa que imprima na tela todos os números pares entre 0 e 100.

19. Construa um programa que peça ao usuário quatro números e imprima-os na tela em ordem crescente.

20. Crie um programa que receba dois números inteiros positivos entre 0 e 1000 e imprima na tela todos os números inteiros que estão no intervalo entre eles.

4.3.1 Respostas dos exercícios



```
exercicio2_1.py
1 numero = input('Digite um numero inteiro: ')
2 if numero<0:
3     print('O numero e negativo.')
4 else:
5     print('O numero e positivo.')
```

piantola@MacBook-Pro ~ % python -u "/Users/piantola/Desktop/Py/modulo2/exercicio2_1.py"

Digite um numero inteiro: 30

O numero e positivo.

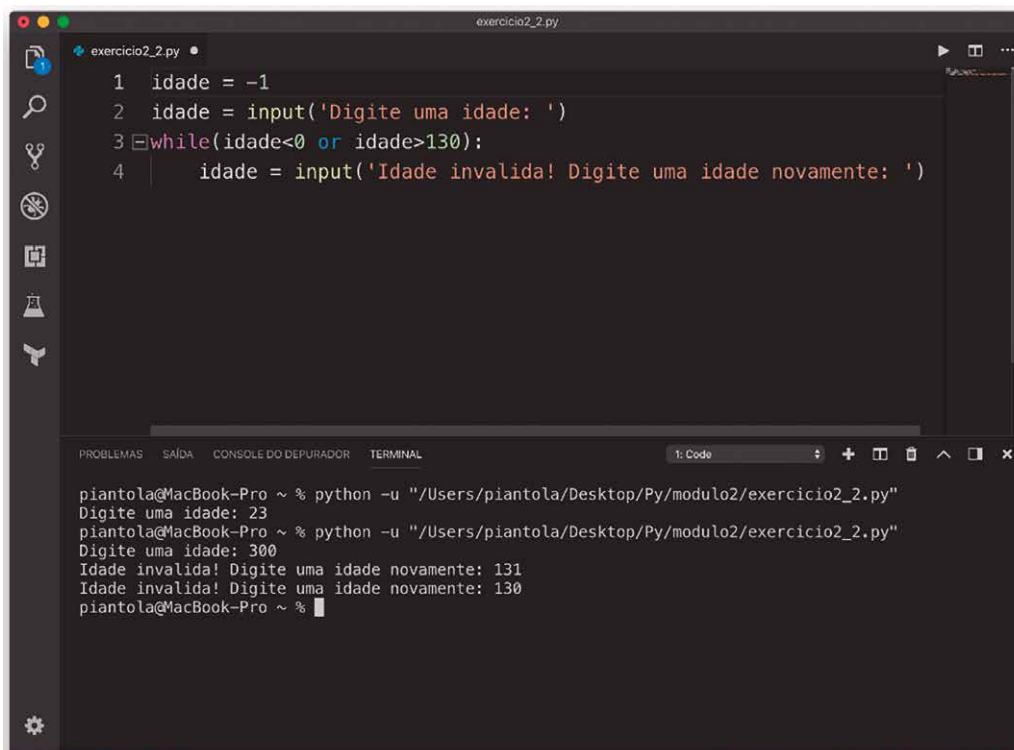
piantola@MacBook-Pro ~ % python -u "/Users/piantola/Desktop/Py/modulo2/exercicio2_1.py"

Digite um numero inteiro: -3

O numero e negativo.

piantola@MacBook-Pro ~ %

Figura 31 – Resolução do exercício 1



```
exercicio2_2.py
1 idade = -1
2 idade = input('Digite uma idade: ')
3 while(idade<0 or idade>130):
4     idade = input('Idade invalida! Digite uma idade novamente: ')
```

piantola@MacBook-Pro ~ % python -u "/Users/piantola/Desktop/Py/modulo2/exercicio2_2.py"

Digite uma idade: 23

piantola@MacBook-Pro ~ % python -u "/Users/piantola/Desktop/Py/modulo2/exercicio2_2.py"

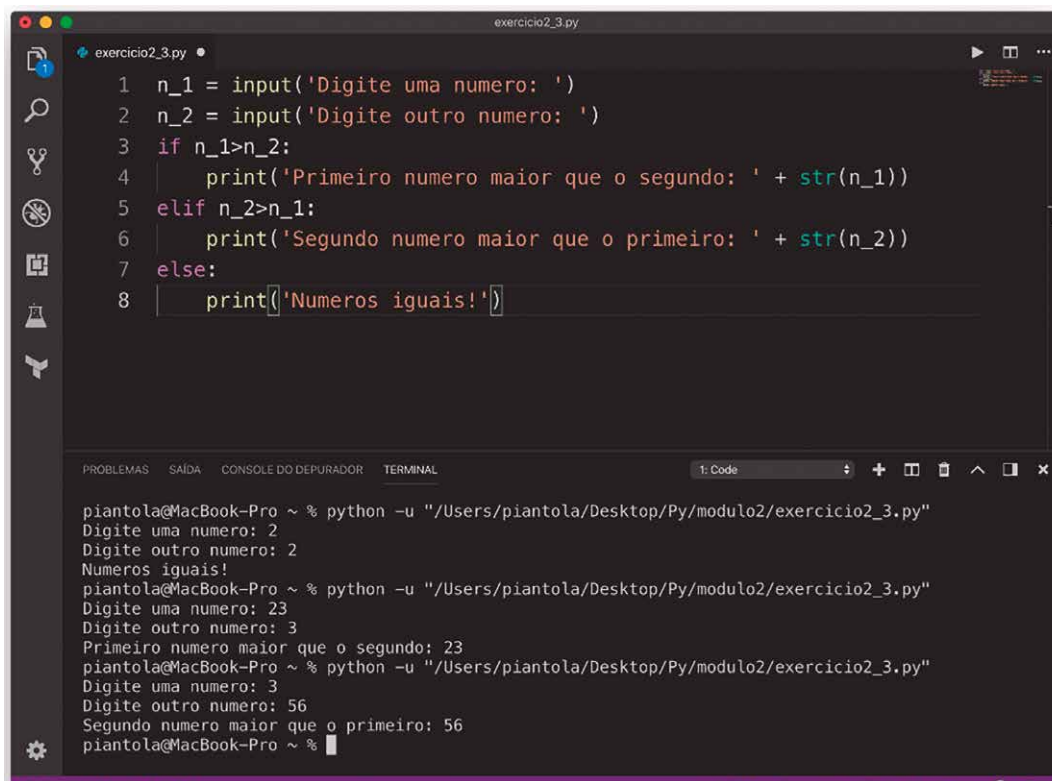
Digite uma idade: 300

Idade invalida! Digite uma idade novamente: 131

Idade invalida! Digite uma idade novamente: 130

piantola@MacBook-Pro ~ %

Figura 32 – Resolução do exercício 2

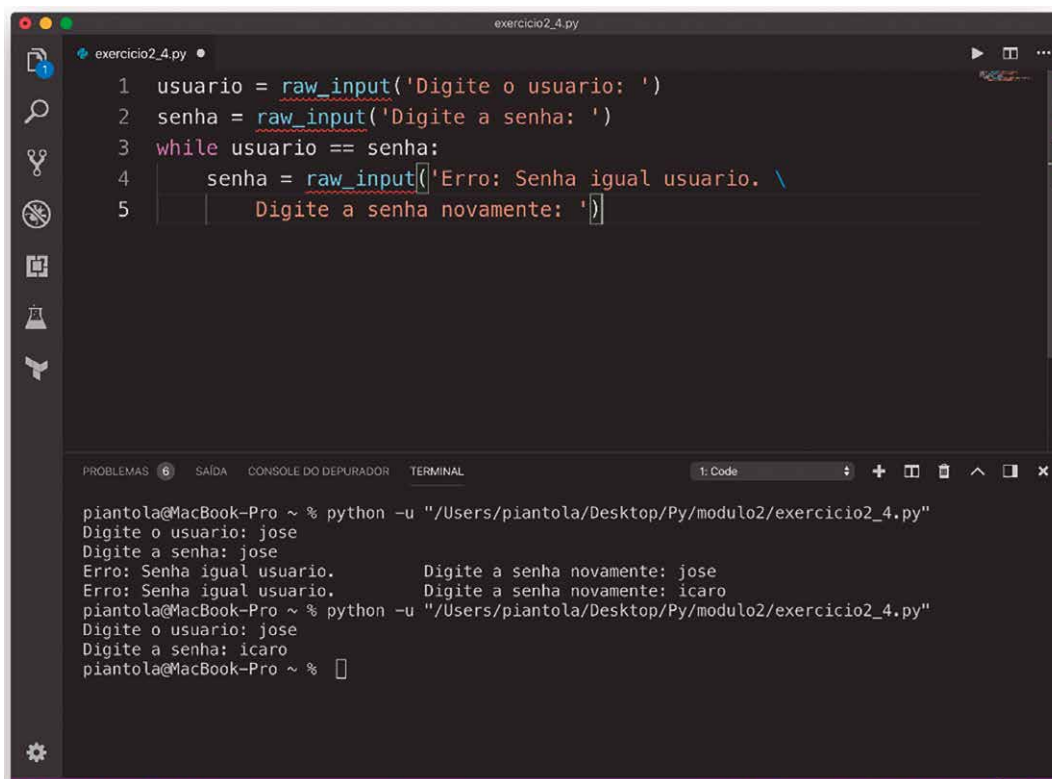


```
exercico2_3.py
1 n_1 = input('Digite uma numero: ')
2 n_2 = input('Digite outro numero: ')
3 if n_1>n_2:
4     print('Primeiro numero maior que o segundo: ' + str(n_1))
5 elif n_2>n_1:
6     print('Segundo numero maior que o primeiro: ' + str(n_2))
7 else:
8     print('Numeros iguais!')
```

PROBLEMAS SAÍDA CONSOLE DO DEPURADOR TERMINAL 1: Code

```
piantola@MacBook-Pro ~ % python -u "/Users/piantola/Desktop/Py/modulo2/exercico2_3.py"
Digite uma numero: 2
Digite outro numero: 2
Numeros iguais!
piantola@MacBook-Pro ~ % python -u "/Users/piantola/Desktop/Py/modulo2/exercico2_3.py"
Digite uma numero: 23
Digite outro numero: 3
Primeiro numero maior que o segundo: 23
piantola@MacBook-Pro ~ % python -u "/Users/piantola/Desktop/Py/modulo2/exercico2_3.py"
Digite uma numero: 3
Digite outro numero: 56
Segundo numero maior que o primeiro: 56
piantola@MacBook-Pro ~ %
```

Figura 33 – Resolução do exercício 3

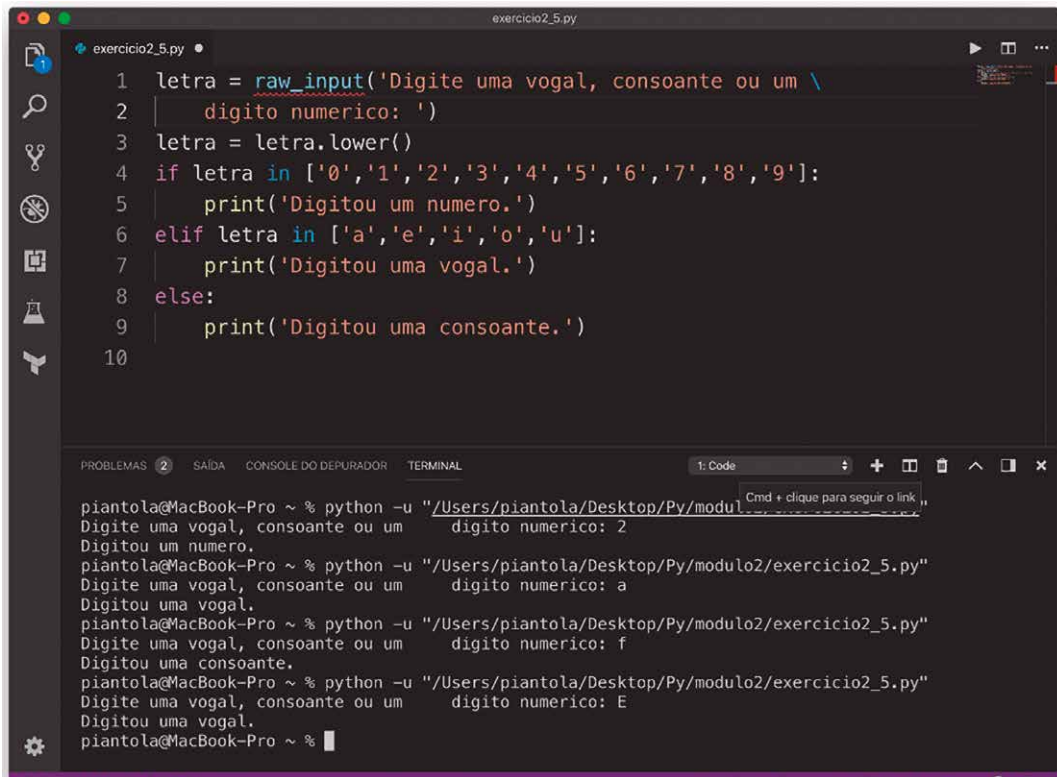


```
exercico2_4.py
1 usuario = raw_input('Digite o usuario: ')
2 senha = raw_input('Digite a senha: ')
3 while usuario == senha:
4     senha = raw_input('Erro: Senha igual usuario. \
5         Digite a senha novamente: ')
```

PROBLEMAS 6 SAÍDA CONSOLE DO DEPURADOR TERMINAL 1: Code

```
piantola@MacBook-Pro ~ % python -u "/Users/piantola/Desktop/Py/modulo2/exercico2_4.py"
Digite o usuario: jose
Digite a senha: jose
Erro: Senha igual usuario. Digite a senha novamente: jose
Erro: Senha igual usuario. Digite a senha novamente: icaro
piantola@MacBook-Pro ~ % python -u "/Users/piantola/Desktop/Py/modulo2/exercico2_4.py"
Digite o usuario: jose
Digite a senha: icaro
piantola@MacBook-Pro ~ %
```

Figura 34 – Resolução do exercício 4

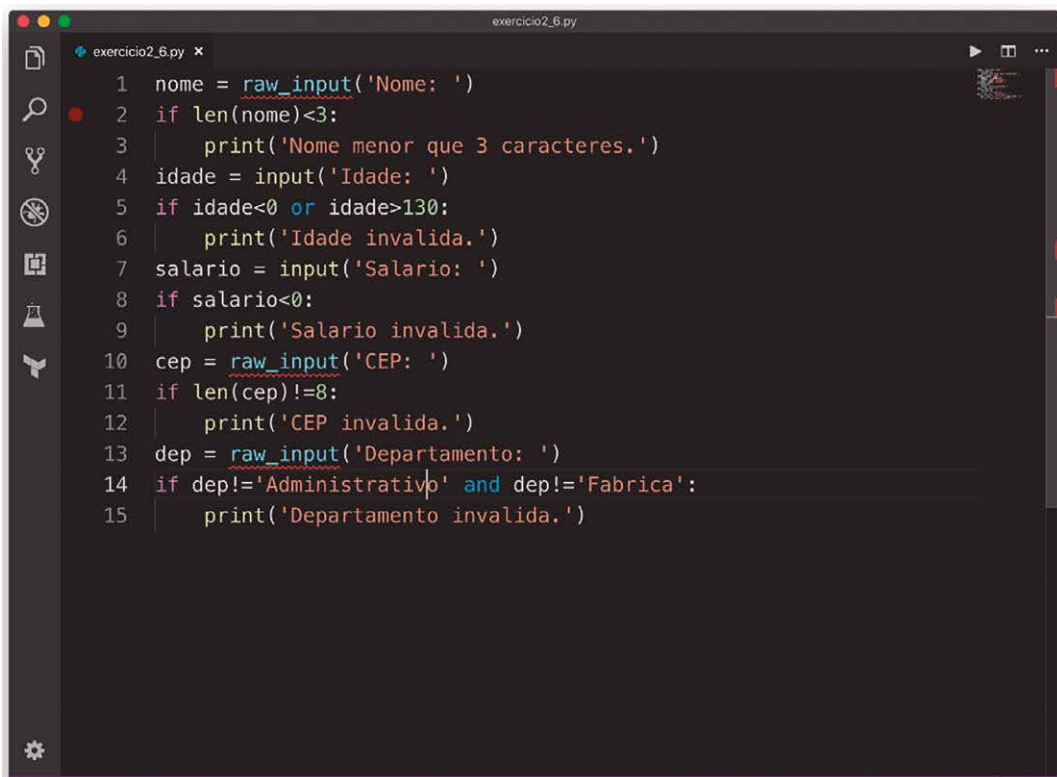


```
exercico2_5.py
1 letra = raw_input('Digite uma vogal, consoante ou um \
2     digito numerico: ')
3 letra = letra.lower()
4 if letra in ['0','1','2','3','4','5','6','7','8','9']:
5     print('Digitou um numero.')
6 elif letra in ['a','e','i','o','u']:
7     print('Digitou uma vogal.')
8 else:
9     print('Digitou uma consoante.')
10
```

PROBLEMAS 2 SAÍDA CONSOLE DO DEPURADOR TERMINAL 1: Code

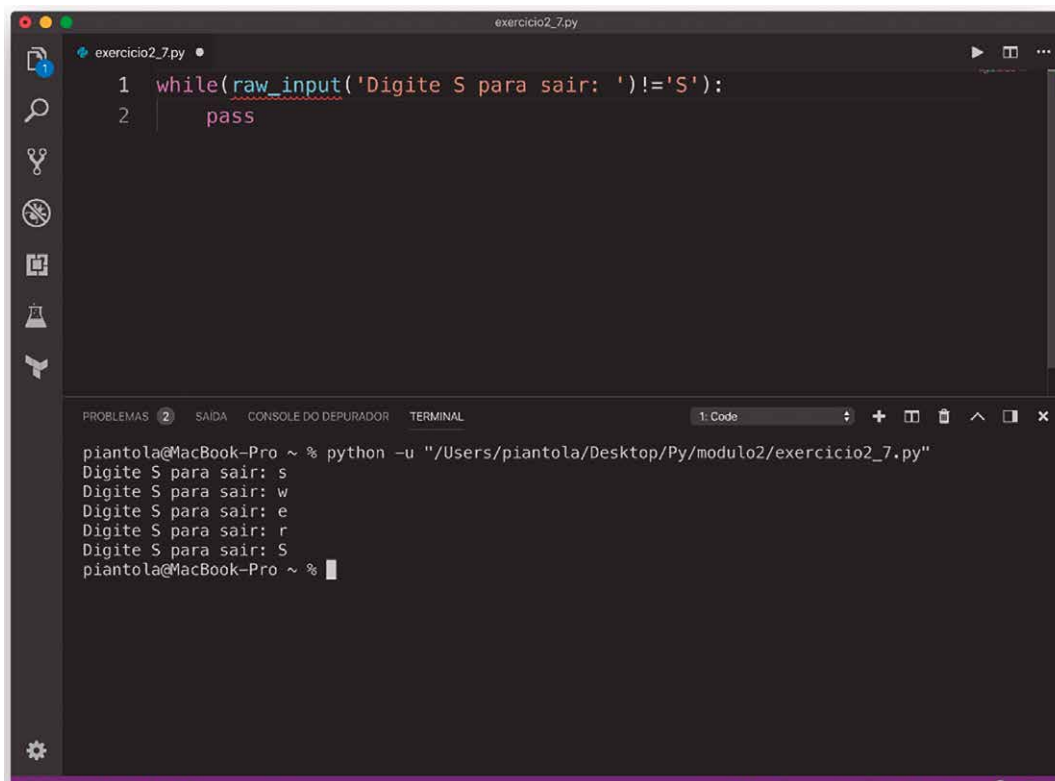
```
piantola@MacBook-Pro ~ % python -u "/Users/piantola/Desktop/Py/modulo2/exercicio2_5.py"
Digite uma vogal, consoante ou um digito numerico: 2
Digitou um numero.
piantola@MacBook-Pro ~ % python -u "/Users/piantola/Desktop/Py/modulo2/exercicio2_5.py"
Digite uma vogal, consoante ou um digito numerico: a
Digitou uma vogal.
piantola@MacBook-Pro ~ % python -u "/Users/piantola/Desktop/Py/modulo2/exercicio2_5.py"
Digite uma vogal, consoante ou um digito numerico: f
Digitou uma consoante.
piantola@MacBook-Pro ~ % python -u "/Users/piantola/Desktop/Py/modulo2/exercicio2_5.py"
Digite uma vogal, consoante ou um digito numerico: E
Digitou uma vogal.
piantola@MacBook-Pro ~ %
```

Figura 35 – Resolução do exercício 5



```
exercico2_6.py
1 nome = raw_input('Nome: ')
2 if len(nome)<3:
3     print('Nome menor que 3 caracteres.')
4 idade = input('Idade: ')
5 if idade<0 or idade>130:
6     print('Idade invalida.')
7 salario = input('Salario: ')
8 if salario<0:
9     print('Salario invalida.')
10 cep = raw_input('CEP: ')
11 if len(cep)!=8:
12     print('CEP invalida.')
13 dep = raw_input('Departamento: ')
14 if dep!='Administrativo' and dep!='Fabrica':
15     print('Departamento invalida.')
```

Figura 36 – Resolução do exercício 6

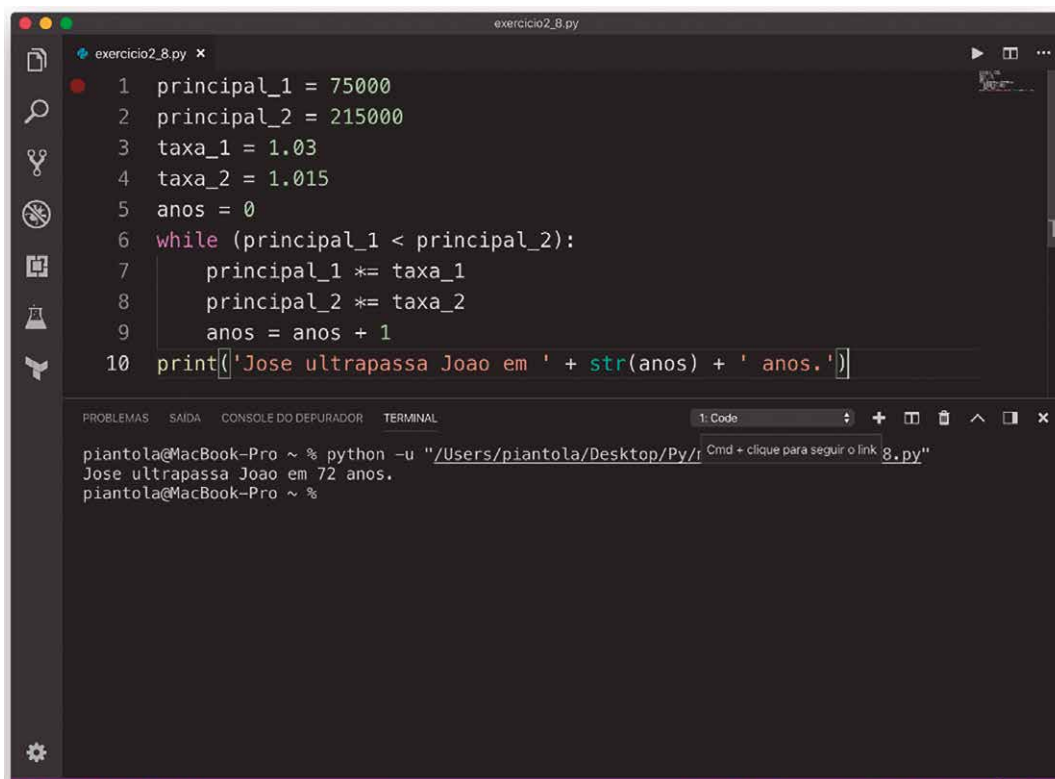


```
exercico2_7.py
1 while(raw_input('Digite S para sair: ')!='S'):
2     pass

PROBLEMAS 2 SAIDA CONSOLE DO DEPURADOR TERMINAL
1: Code

piantola@MacBook-Pro ~ % python -u "/Users/piantola/Desktop/Py/modulo2/exercico2_7.py"
Digite S para sair: s
Digite S para sair: w
Digite S para sair: e
Digite S para sair: r
Digite S para sair: S
piantola@MacBook-Pro ~ %
```

Figura 37 – Resolução do exercício 7

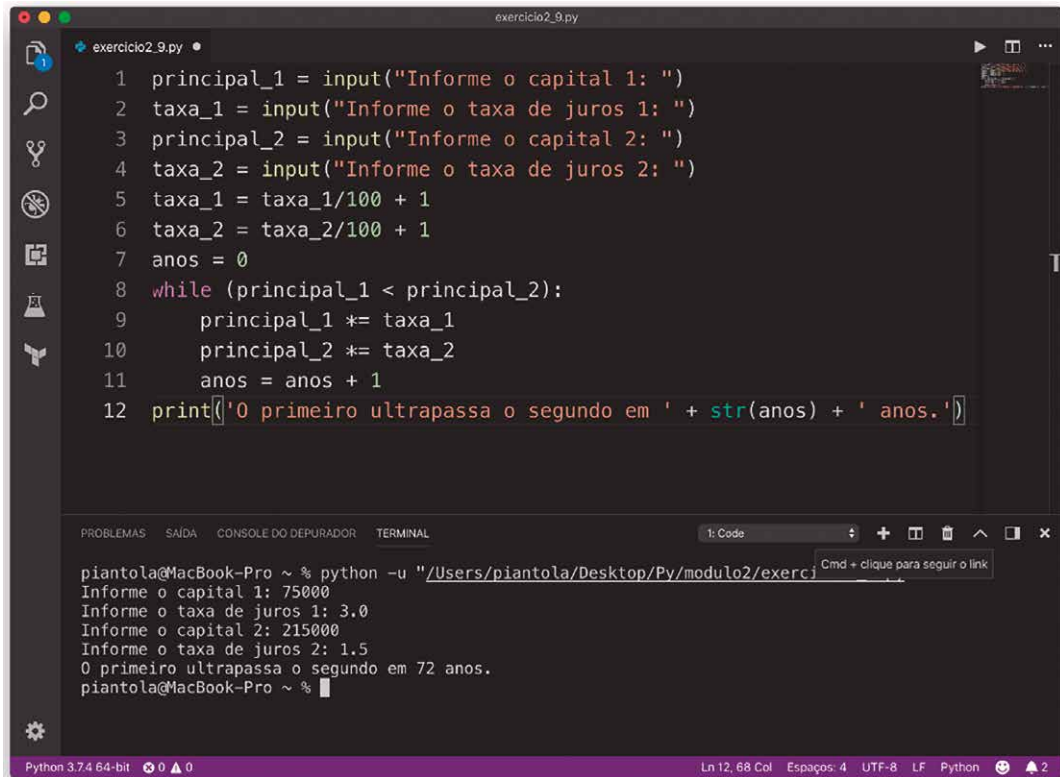


```
exercico2_8.py
1 principal_1 = 75000
2 principal_2 = 215000
3 taxa_1 = 1.03
4 taxa_2 = 1.015
5 anos = 0
6 while (principal_1 < principal_2):
7     principal_1 *= taxa_1
8     principal_2 *= taxa_2
9     anos = anos + 1
10 print('Jose ultrapassa Joao em ' + str(anos) + ' anos.')
```

```
piantola@MacBook-Pro ~ % python -u "/Users/piantola/Desktop/Py/i Cmd + clique para seguir o link 8.py"
Jose ultrapassa Joao em 72 anos.
piantola@MacBook-Pro ~ %
```

Figura 38 – Resolução do exercício 8

INTRODUÇÃO À PROGRAMAÇÃO ESTRUTURADA



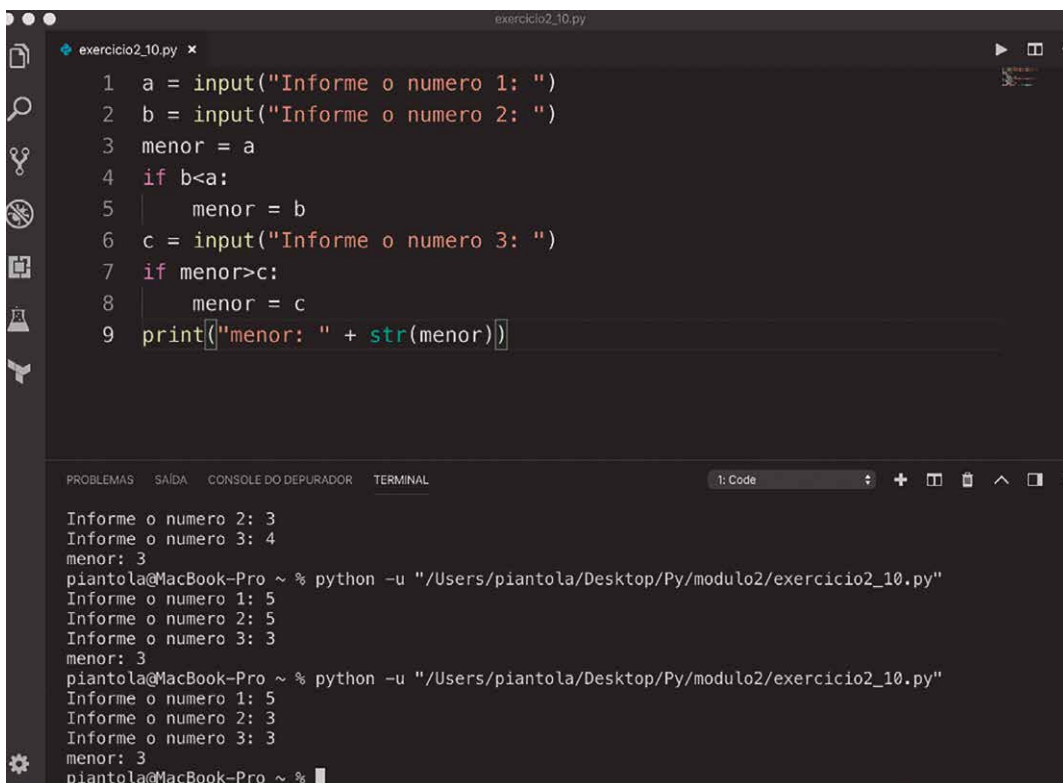
```
exercício2_9.py
1 principal_1 = input("Informe o capital 1: ")
2 taxa_1 = input("Informe o taxa de juros 1: ")
3 principal_2 = input("Informe o capital 2: ")
4 taxa_2 = input("Informe o taxa de juros 2: ")
5 taxa_1 = taxa_1/100 + 1
6 taxa_2 = taxa_2/100 + 1
7 anos = 0
8 while (principal_1 < principal_2):
9     principal_1 *= taxa_1
10    principal_2 *= taxa_2
11    anos = anos + 1
12 print('0 primeiro ultrapassa o segundo em ' + str(anos) + ' anos.')
```

PROBLEMAS SAÍDA CONSOLE DO DEPURADOR TERMINAL

```
piantola@MacBook-Pro ~ % python -u "/Users/piantola/Desktop/Py/modulo2/exercício2_9.py"
Informe o capital 1: 75000
Informe o taxa de juros 1: 3.0
Informe o capital 2: 215000
Informe o taxa de juros 2: 1.5
0 primeiro ultrapassa o segundo em 72 anos.
piantola@MacBook-Pro ~ %
```

Python 3.7.4 64-bit 0 0 0 Ln 12, 68 Col Espaços: 4 UTF-8 LF Python 2

Figura 39 – Resolução do exercício 9

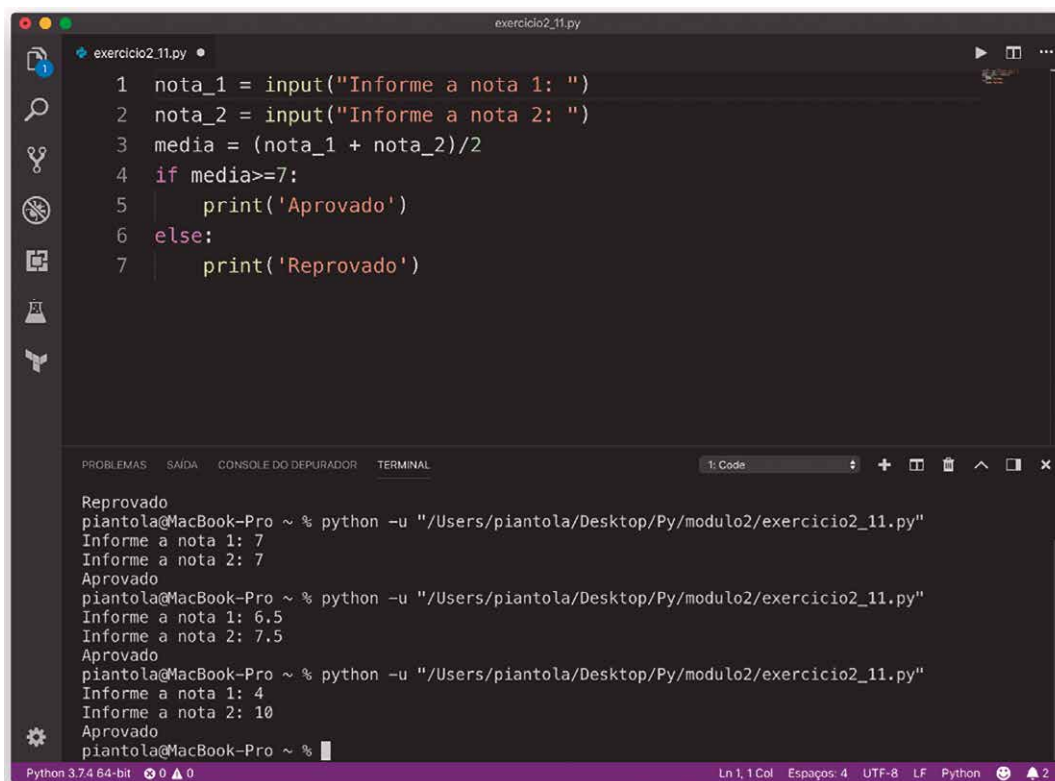


```
exercício2_10.py
1 a = input("Informe o numero 1: ")
2 b = input("Informe o numero 2: ")
3 menor = a
4 if b < a:
5     menor = b
6 c = input("Informe o numero 3: ")
7 if menor > c:
8     menor = c
9 print("menor: " + str(menor))
```

PROBLEMAS SAÍDA CONSOLE DO DEPURADOR TERMINAL

```
Informe o numero 2: 3
Informe o numero 3: 4
menor: 3
piantola@MacBook-Pro ~ % python -u "/Users/piantola/Desktop/Py/modulo2/exercício2_10.py"
Informe o numero 1: 5
Informe o numero 2: 5
Informe o numero 3: 3
menor: 3
piantola@MacBook-Pro ~ % python -u "/Users/piantola/Desktop/Py/modulo2/exercício2_10.py"
Informe o numero 1: 5
Informe o numero 2: 3
Informe o numero 3: 3
menor: 3
piantola@MacBook-Pro ~ %
```

Figura 40 – Resolução do exercício 10



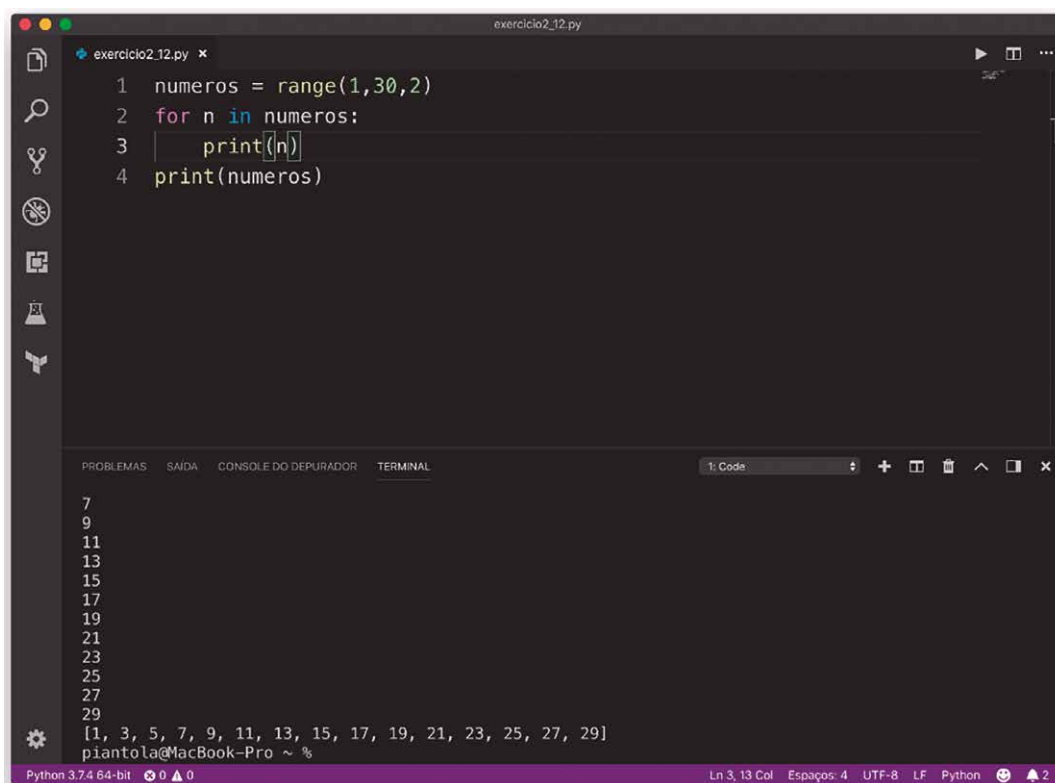
```
exercicio2_11.py
1 nota_1 = input("Informe a nota 1: ")
2 nota_2 = input("Informe a nota 2: ")
3 media = (nota_1 + nota_2)/2
4 if media >= 7:
5     print('Aprovado')
6 else:
7     print('Reprovado')
```

PROBLEMAS SAÍDA CONSOLE DO DEPURADOR TERMINAL 1: Code

```
Reprovado
piantola@MacBook-Pro ~ % python -u "/Users/piantola/Desktop/Py/modulo2/exercicio2_11.py"
Informe a nota 1: 7
Informe a nota 2: 7
Aprovado
piantola@MacBook-Pro ~ % python -u "/Users/piantola/Desktop/Py/modulo2/exercicio2_11.py"
Informe a nota 1: 6,5
Informe a nota 2: 7,5
Aprovado
piantola@MacBook-Pro ~ % python -u "/Users/piantola/Desktop/Py/modulo2/exercicio2_11.py"
Informe a nota 1: 4
Informe a nota 2: 10
Aprovado
piantola@MacBook-Pro ~ %
```

Python 3.7.4 64-bit 0 0 Ln 1, 1 Col Espaços: 4 UTF-8 LF Python 2

Figura 41 – Resolução do exercício 11



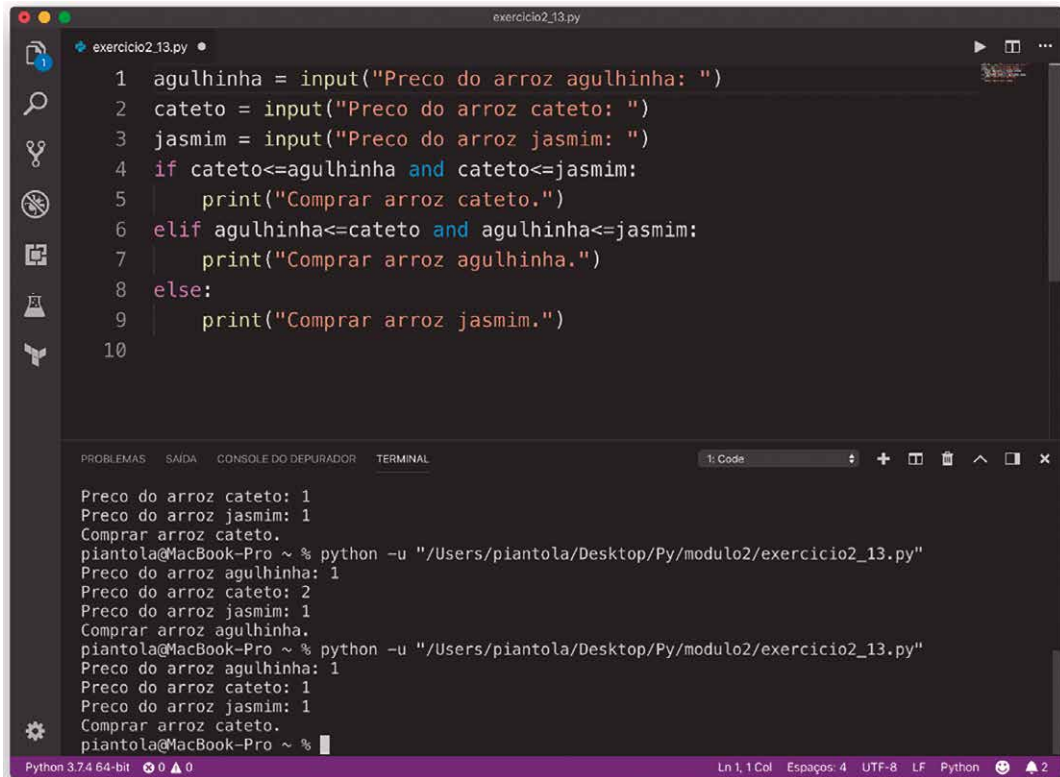
```
exercicio2_12.py
1 numeros = range(1,30,2)
2 for n in numeros:
3     print(n)
4 print(numeros)
```

PROBLEMAS SAÍDA CONSOLE DO DEPURADOR TERMINAL 1: Code

```
7
9
11
13
15
17
19
21
23
25
27
29
[1, 3, 5, 7, 9, 11, 13, 15, 17, 19, 21, 23, 25, 27, 29]
piantola@MacBook-Pro ~ %
```

Python 3.7.4 64-bit 0 0 Ln 3, 13 Col Espaços: 4 UTF-8 LF Python 2

Figura 42 – Resolução do exercício 12

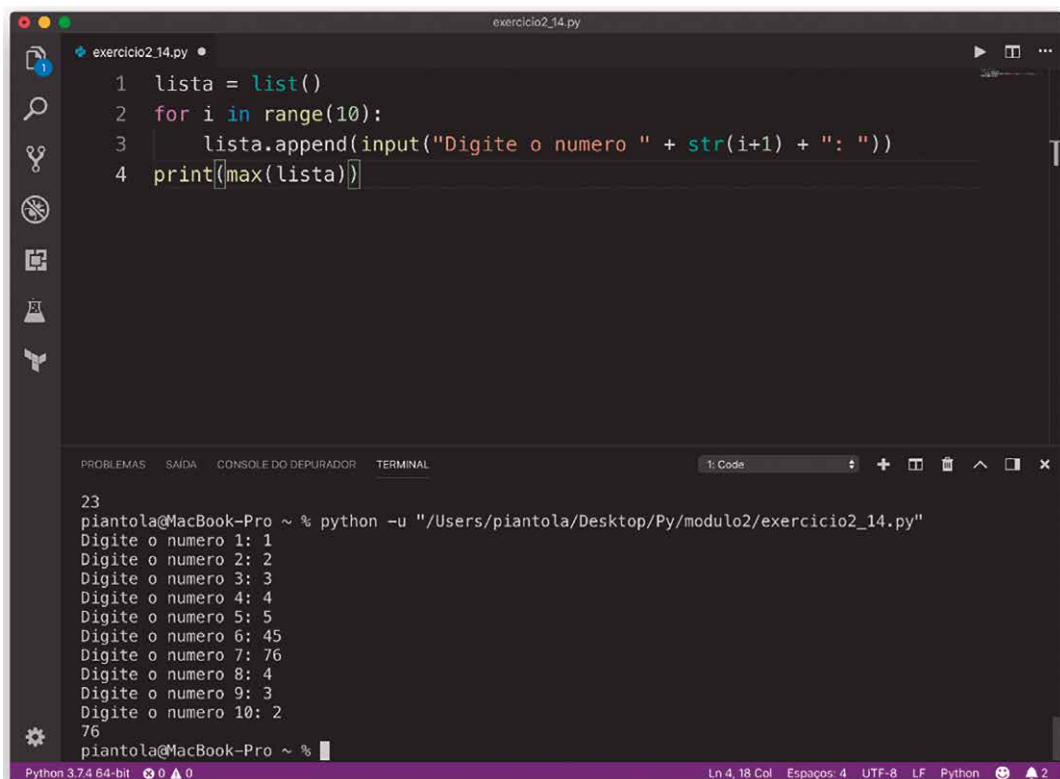


```
exercicio2_13.py
1 agulhinha = input("Preço do arroz agulhinha: ")
2 cateto = input("Preço do arroz cateto: ")
3 jasmim = input("Preço do arroz jasmim: ")
4 if cateto <= agulhinha and cateto <= jasmim:
5     print("Comprar arroz cateto.")
6 elif agulhinha <= cateto and agulhinha <= jasmim:
7     print("Comprar arroz agulhinha.")
8 else:
9     print("Comprar arroz jasmim.")
10

PROBLEMAS SAÍDA CONSOLA DO DEPURADOR TERMINAL
1: Code

Preço do arroz cateto: 1
Preço do arroz jasmim: 1
Comprar arroz cateto.
piantola@MacBook-Pro ~ % python -u "/Users/piantola/Desktop/Py/modulo2/exercicio2_13.py"
Preço do arroz agulhinha: 1
Preço do arroz cateto: 2
Preço do arroz jasmim: 1
Comprar arroz agulhinha.
piantola@MacBook-Pro ~ % python -u "/Users/piantola/Desktop/Py/modulo2/exercicio2_13.py"
Preço do arroz agulhinha: 1
Preço do arroz cateto: 1
Preço do arroz jasmim: 1
Comprar arroz cateto.
piantola@MacBook-Pro ~ %
```

Figura 43 – Resolução do exercício 13

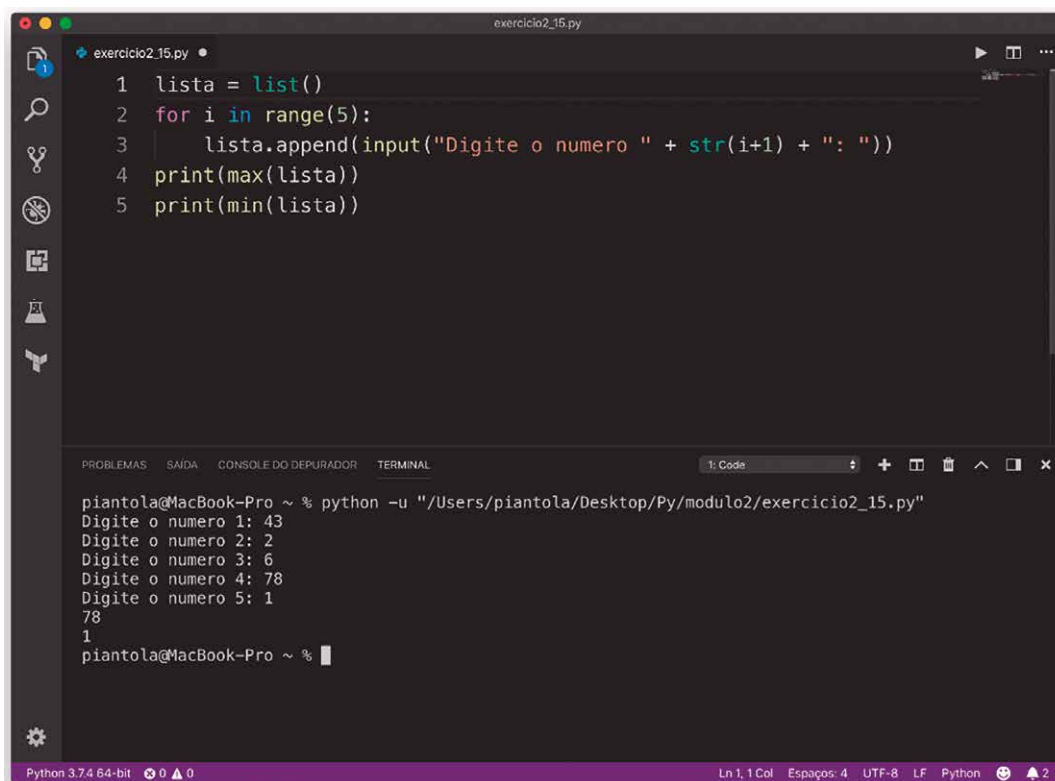


```
exercicio2_14.py
1 lista = list()
2 for i in range(10):
3     lista.append(input("Digite o numero " + str(i+1) + ": "))
4 print(max(lista))

PROBLEMAS SAÍDA CONSOLA DO DEPURADOR TERMINAL
1: Code

23
piantola@MacBook-Pro ~ % python -u "/Users/piantola/Desktop/Py/modulo2/exercicio2_14.py"
Digite o numero 1: 1
Digite o numero 2: 2
Digite o numero 3: 3
Digite o numero 4: 4
Digite o numero 5: 5
Digite o numero 6: 45
Digite o numero 7: 76
Digite o numero 8: 4
Digite o numero 9: 3
Digite o numero 10: 2
76
piantola@MacBook-Pro ~ %
```

Figura 44 – Resolução do exercício 14



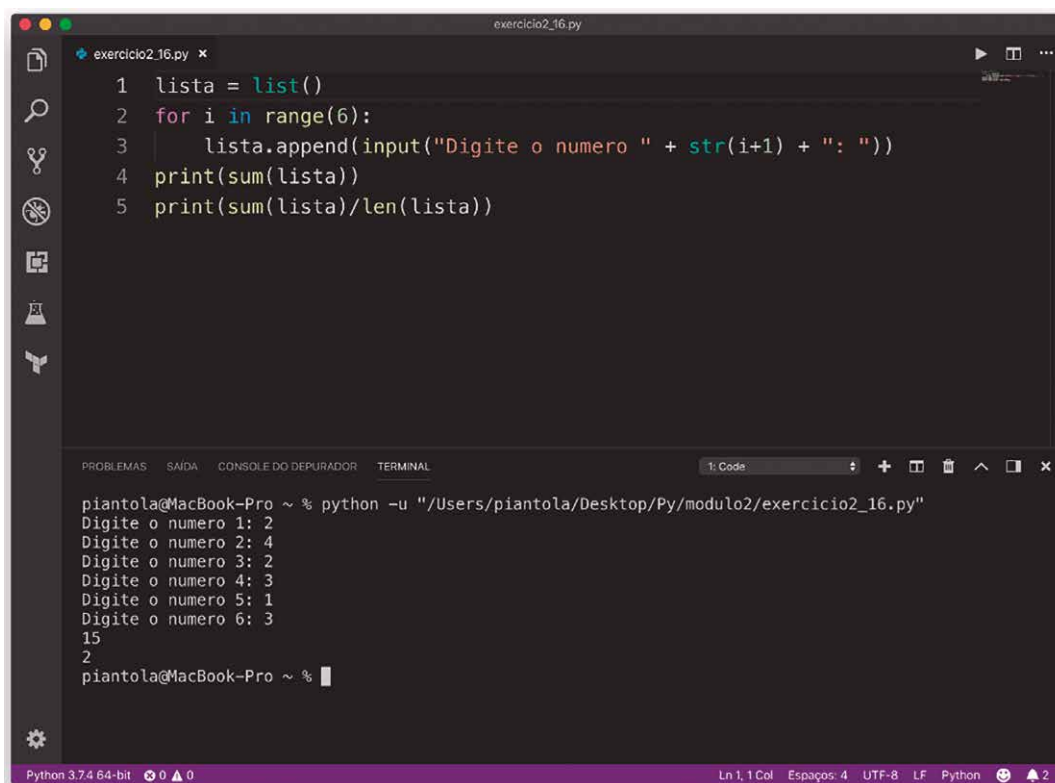
```
exercicio2_15.py
1 lista = list()
2 for i in range(5):
3     lista.append(input("Digite o numero " + str(i+1) + ": "))
4 print(max(lista))
5 print(min(lista))
```

piantola@MacBook-Pro ~ % python -u "/Users/piantola/Desktop/Py/modulo2/exercicio2_15.py"

Digite o numero 1: 43
Digite o numero 2: 2
Digite o numero 3: 6
Digite o numero 4: 78
Digite o numero 5: 1
78
1
piantola@MacBook-Pro ~ %

Python 3.7.4 64-bit 0 0 Ln 1, 1 Col Espaços: 4 UTF-8 LF Python 2

Figura 45 – Resolução do exercício 15



```
exercicio2_16.py
1 lista = list()
2 for i in range(6):
3     lista.append(input("Digite o numero " + str(i+1) + ": "))
4 print(sum(lista))
5 print(sum(lista)/len(lista))
```

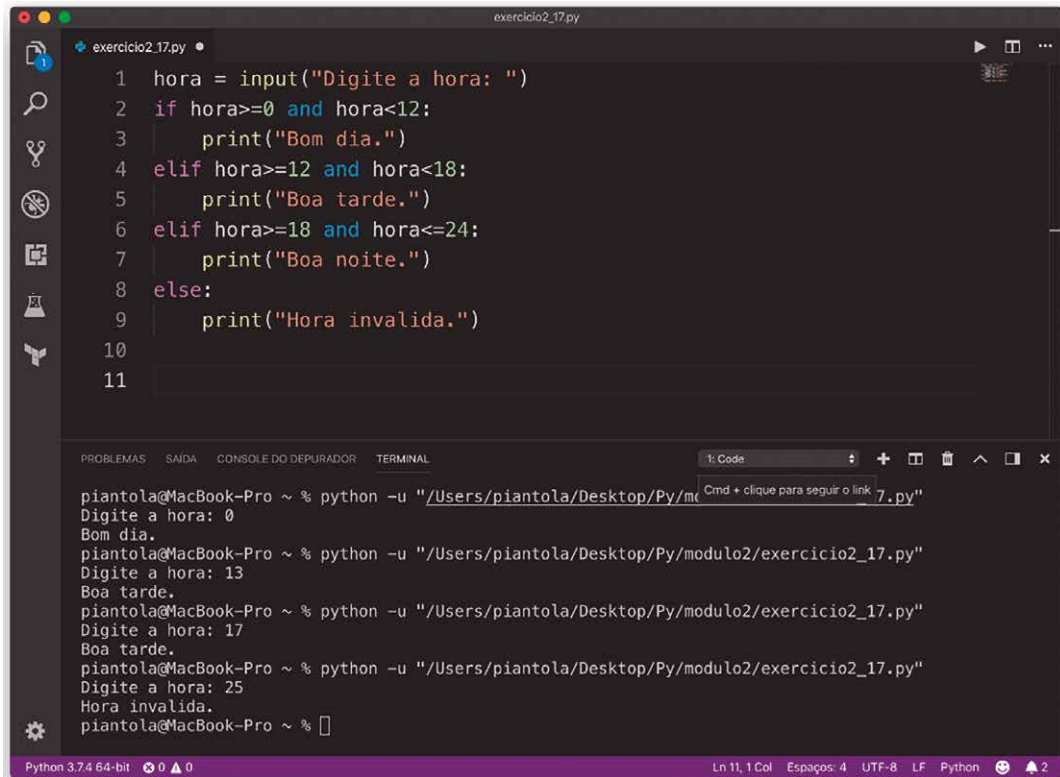
piantola@MacBook-Pro ~ % python -u "/Users/piantola/Desktop/Py/modulo2/exercicio2_16.py"

Digite o numero 1: 2
Digite o numero 2: 4
Digite o numero 3: 2
Digite o numero 4: 3
Digite o numero 5: 1
Digite o numero 6: 3
15
2
piantola@MacBook-Pro ~ %

Python 3.7.4 64-bit 0 0 Ln 1, 1 Col Espaços: 4 UTF-8 LF Python 2

Figura 46 – Resolução do exercício 16

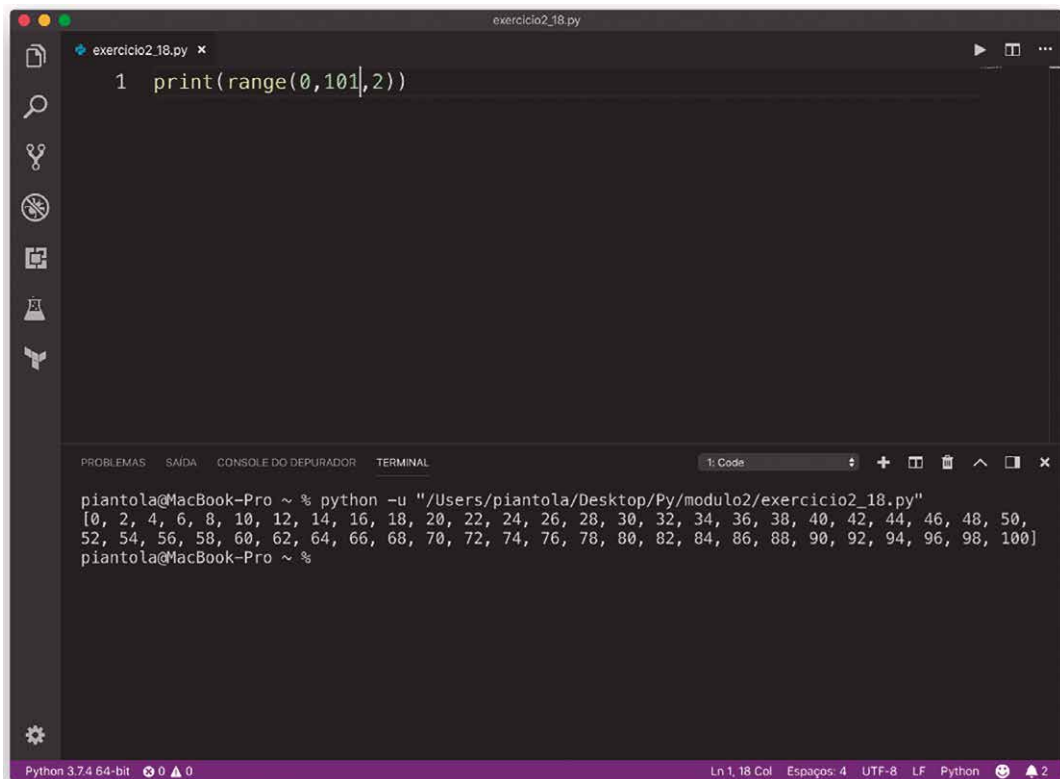
INTRODUÇÃO À PROGRAMAÇÃO ESTRUTURADA



```
exercicio2_17.py
1 hora = input("Digite a hora: ")
2 if hora>=0 and hora<12:
3     print("Bom dia.")
4 elif hora>=12 and hora<18:
5     print("Boa tarde.")
6 elif hora>=18 and hora<=24:
7     print("Boa noite.")
8 else:
9     print("Hora invalida.")
10
11

PROBLEMAS SAÍDA CONSOLE DO DEPURADOR TERMINAL
1: Code
piantola@MacBook-Pro ~ % python -u "/Users/piantola/Desktop/Py/modulo2/exercicio2_17.py"
Digite a hora: 0
Bom dia.
piantola@MacBook-Pro ~ % python -u "/Users/piantola/Desktop/Py/modulo2/exercicio2_17.py"
Digite a hora: 13
Boa tarde.
piantola@MacBook-Pro ~ % python -u "/Users/piantola/Desktop/Py/modulo2/exercicio2_17.py"
Digite a hora: 17
Boa tarde.
piantola@MacBook-Pro ~ % python -u "/Users/piantola/Desktop/Py/modulo2/exercicio2_17.py"
Digite a hora: 25
Hora invalida.
piantola@MacBook-Pro ~ %
```

Figura 47 – Resolução do exercício 17



```
exercicio2_18.py
1 print(range(0,101,2))

PROBLEMAS SAÍDA CONSOLE DO DEPURADOR TERMINAL
1: Code
piantola@MacBook-Pro ~ % python -u "/Users/piantola/Desktop/Py/modulo2/exercicio2_18.py"
[0, 2, 4, 6, 8, 10, 12, 14, 16, 18, 20, 22, 24, 26, 28, 30, 32, 34, 36, 38, 40, 42, 44, 46, 48, 50, 52, 54, 56, 58, 60, 62, 64, 66, 68, 70, 72, 74, 76, 78, 80, 82, 84, 86, 88, 90, 92, 94, 96, 98, 100]
piantola@MacBook-Pro ~ %
```

Figura 48 – Resolução do exercício 18

```
exercicio2_19.py
1 lista = list()
2 for i in range(4):
3     lista.append(input("Digite um numero " + str(i+1) + ": "))
4 lista.sort()
5 print(lista)

python -u "/Users/piantola/Desktop/Py/modulo2/exer... Cmd + clique para seguir o link
piantola@MacBook-Pro ~ % python -u "/Users/piantola/Desktop/Py/modulo2/exercicio2_19.py"
Digite um numero 1: 4
Digite um numero 2: 2
Digite um numero 3: 3
Digite um numero 4: 1
[1, 2, 3, 4]
piantola@MacBook-Pro ~ %
```

Figura 49 – Resolução do exercício 19

```
exercicio2_20.py
1 a = input("Digite o menor numero 1: (0-1000)")
2 b = input("Digite o maior numero 2: (0-1000)")
3 if a<0 or a>1000 or b<0 or b>1000:
4     print("Numeros invalidos.")
5 else:
6     print(range(a,b+1,1))

piantola@MacBook-Pro ~ % python -u "/Users/piantola/Desktop/Py/modulo2/exercicio2_20.py"
Digite o menor numero 1: (0-1000)140
Digite o maior numero 2: (0-1000)213
[140, 141, 142, 143, 144, 145, 146, 147, 148, 149, 150, 151, 152, 153, 154, 155, 156, 157, 158, 159,
160, 161, 162, 163, 164, 165, 166, 167, 168, 169, 170, 171, 172, 173, 174, 175, 176, 177, 178, 179,
180, 181, 182, 183, 184, 185, 186, 187, 188, 189, 190, 191, 192, 193, 194, 195, 196, 197, 198, 199,
200, 201, 202, 203, 204, 205, 206, 207, 208, 209, 210, 211, 212, 213]
piantola@MacBook-Pro ~ %
```

Figura 50 – Resolução do exercício 20



Resumo

Nesta unidade, foi estudada a estrutura condicional, também conhecida como desvio condicional ou decisão. Ela é utilizada para decidir se um conjunto de instruções ou comandos deve ou não ser executado.

Em Python, as instruções responsáveis pela estrutura de decisão são: `if`, `if-else` e `elif`. A instrução `if` testa uma expressão lógica e, caso ela seja verdadeira, será executado um conjunto de comandos logo abaixo do `if`. No caso de a expressão lógica ser falsa, o fluxo do programa segue a sequência sem executar o bloco `if`; se na sequência estiver o bloco `else`, ele será executado. O `elif` tem uma segunda expressão lógica a ser testada caso a expressão do `if` seja falsa. É possível ter vários `elif` na estrutura, porém somente um único `else`. Também é possível colocar decisões em cadeias, usando uma instrução `if` dentro de outra.

É importante ressaltar que os blocos de instruções são separados por indentação. A indentação geralmente são quatro espaços, mas pode ser qualquer quantidade de espaços ou tabulações, desde que todo o programa mantenha um padrão na quantidade escolhida. Caso um bloco `if` não tenha nenhuma instrução, ele não pode ficar vazio. Nesse caso, utilizamos a instrução `pass`, que não faz nada.

O `switch-case` é outra estrutura condicional utilizada em várias linguagens de programação. Diferente do `if-elif-else`, ele não compara uma expressão booleana, mas sim o valor de uma variável. Essa estrutura não existe em Python, mas a que existe é suficiente para os fins de desenvolvimento de software.

As estruturas de repetição permitem executar um bloco de comandos repetidamente até que uma condição definida pelo programador seja atingida ou não seja mais verdadeira. Essas estruturas são conhecidas também como estruturas de laço.

Em Python, as principais estruturas de repetição são o `while` e o `for`. O `while` testa uma expressão lógica; se ela for verdadeira, o bloco do `while` é executado. O laço `for` é utilizado na linguagem Python para interagir com uma estrutura de dados sequencial, que pode ser uma lista, um dicionário, um conjunto, uma tupla ou uma string. É possível usar estrutura encadeada para laços também, ou seja, colocar comandos `while` ou `for`, dentro deles mesmos.

As instruções `break` e `continue` podem ser usadas dentro das estruturas de laços como `while` e `for` para, respectivamente, sair do laço completamente e ir para próxima iteração imediatamente.



Exercícios

Questão 1. Para facilitar a escrita de códigos com if-else aninhados, a linguagem Python oferece as estruturas de condição if-elif-else. O comando elif, que pode ser entendido como uma contração de else if, torna mais claro o tratamento de várias condições do mesmo contexto. Blocos de elif podem ser repetidos diversas vezes, quando necessário.

Com base nessas informações, avalie o código a seguir:

```
nota1 = float(input("Digite a nota da 1ª prova: "))
nota2 = float(input("Digite a nota da 2ª prova: "))

media = (nota1 + 2*nota2)/3

if media < 5:
    print("Reprovado.")
elif media < 6:
    print("Recuperação.")
elif media < 9:
    print("Aprovado. Parabéns!")
else:
    print("Aprovado com louvor! Parabéns!")
```

Durante a execução, o usuário informou que as notas da primeira e da segunda prova são, respectivamente, 9.6 e 8.7. Nessa condição, qual será a saída esperada para o programa?

- A) Aprovado com louvor! Parabéns!
- B) Aprovado. Parabéns!
- C) Recuperação.
- D) Reprovado.
- E) Sua média é igual a 8.7.

Resposta correta: alternativa A.

Análise da questão

O programa solicita ao usuário, por meio do comando input(), as notas de duas provas. O comando float() converte os dados recebidos para o tipo ponto flutuante. Dessa forma, é possível realizar operações aritméticas com eles. Com isso, é calculada a média em formato ponderado: a nota1 tem peso 1 e a nota2 tem peso 2, sendo que o somatório das notas é posteriormente dividido por 3 (que representa o somatório dos pesos).

Depois, deparamos com diversos comandos condicionais, que testam o valor da variável media. Se a média for um valor menor do que 5, o código imprimirá Reprovado., e a execução do código é

finalizada. Caso a condição do bloco if seja falsa, o programa testa se o valor da média é menor do que 6, e assim por diante.

No contexto da questão, o usuário informou o valor 9.6 para nota1 e 8.7 para nota2. O cálculo da variável media ocorrerá, portanto, desta forma: $(9.6 + (2 \times 8.7)) / 3 = 9$.

Nessa situação, a condição testada pelo comando if é falsa, pois 9 não é menor do que 5. O mesmo ocorrerá para o primeiro elif, pois 9 também não é menor do que 6. Quando passarmos para o segundo elif, também teremos uma condição falsa, pois 9 não é menor do que 9 (e, sim, igual). Logo, o comando print() que será, de fato, executado é o do bloco do else. Portanto, espera-se ler a impressão Aprovado com louvor! Parabéns! no output da execução.

É importante destacar que nem todo interpretador on-line trabalha adequadamente com o comando input(). Se você encontrou problemas para executar esse código, pesquise por um interpretador que tenha suporte ao comando, ou utilize outra ferramenta, como o Visual Studio Code.

Questão 2. Em Python, o laço de repetição for pode ser utilizado para percorrer os itens de uma coleção de dados, como uma lista, utilizando a sintaxe a seguir:

```
lista = [item1, item2, item]
for variavel in lista:
    comandos
```

Enquanto percorrermos a lista, a variável indicada no laço for recebe, a cada iteração, um item da coleção em questão. Considerando isso, escolha a alternativa que apresenta um exemplo adequado sobre o uso do laço for para percorrer uma lista.

- A)

```
for x = ["John", "Sophie", "Junior"]:
    print(x)
```
- B)

```
array (["John"],["Sophie"],["Junior"])
for each i = 0 to array[i]
    print (array[i])
```
- C)

```
names = ["John", "Sophie", "Junior"]
for x in names:
    print(x)
```
- D)

```
names = { {"John"}, {"Sophie"}, {"Junior"} }
for x in names:
    print(names)
```
- E)

```
array.names("John", "Sophie", "Junior")
for x print(names[x]):
    end for:
```

Resposta correta: alternativa C.

Análise das alternativas

A) Alternativa incorreta.

Justificativa: nessa sintaxe, não é criada uma lista que antecede o laço for. A sintaxe do laço também não corresponde à apresentada no enunciado. A execução do código resulta em erro.

B) Alternativa incorreta.

Justificativa: o comando array não é utilizado para criação de listas em Python. Para criar listas, devemos escolher um nome para ela e atribuir, dentro de colchetes, itens que estarão separados entre si por vírgulas. A sintaxe do laço for também está incorreta. A execução do código resulta em erro.

C) Alternativa correta.

Justificativa: inicialmente, é criada uma lista names, cujos itens são três strings. A variável x, utilizada na estrutura do laço for, permite que seja possível atribuir a ela as strings da lista. A cada loop do for, x assumirá um valor distinto, de modo que seja possível receber todos os elementos da lista names, um de cada vez. O comando dado dentro do bloco do for é o print, de forma que cada item da lista deve ser impresso. Dessa maneira, a saída da execução é a que segue.

John
Sophie
Junior

D) Alternativa incorreta.

Justificativa: quando atribuímos elementos dentro de chaves, estamos tentando criar um conjunto (set), e não uma lista. Esse é outro tipo de estrutura de dados válida em Python. No entanto, os elementos não devem estar envolvidos por chaves dentro das chaves principais (como se estivéssemos tentando criar subconjuntos dentro de um conjunto universo). Dessa forma, também temos um código de sintaxe inválida, que resulta em erro ao ser executado.

E) Alternativa incorreta.

Justificativa: o comando array não é utilizado para criação de listas em Python. A sintaxe do laço for também está incorreta. A execução do código resulta em erro.
