Práticas Computacionais I (v0.0.1)

Daniel R. Cassar

November 26, 2021

Conteúdo

1 Introdução

Você está lendo o caderno de práticas da disciplina Práticas Computacionais I. Como o próprio nome sugere, aqui você encontrará as práticas que serão realizadas durante a disciplina Práticas Computacionais I da Ilum. As práticas estão apresentadas na ordem de construção do conhecimento e é altamente recomendado realizá-las em sequência sem pular nenhuma etapa.

A maioria das práticas contém um breve texto introdutório e uma ou mais questões. As questões são marcadas com uma letra em negrito e entre chaves. **[A]** Por exemplo, aqui seria o enunciado da questão **A** se estivéssemos dentro do escopo de uma prática.

Realize as práticas seguindo as instruções discutidas em sala de aula. Lembre-se que a capacidade de ler, entender, escrever, e depurar seu próprio código de computador é *fundamental* para realizar diversas disciplinas da

Ilum (incluindo, obviamente, todas as disciplinas de computação). Por este motivo, se atente às dicas abaixo:

- A maioria das práticas são individuais. Tente resolver as questões no seu computador antes de discutir em grupo. Depois de tentar resolver, fique à vontade para discutir em grupo e observar outras formas de resolver o mesmo problema;
- Algumas práticas te convidam a executar algum código pronto para ver o que acontece. Mesmo sendo possível copiar e colar o código, é fortemente recomendado *escrever* o código pois aprendemos muito mais quando escrevemos do que quando copiamos e colamos;
- Praticamente todos os problemas computacionais que serão explorados nesta disciplina já foram resolvidos por alguém e a solução pode ser encontrada em livros ou na internet. Isso é inevitável para problemas *simples*. Mais adiante no curso (ou na vida) iremos nos deparar com problemas mais *difíceis* e mais específicos, que não têm soluções prontas. Para resolver problemas difícies precisamos ter uma base sólida que só adquirimos resolvendo problemas mais simples. Aproveite esse momento de aprendizado e evite buscar respostas prontas *antes* de tentar resolver o problema! Depois que você tentou resolver, fique à vontade para procurar e estudar outras soluções.

Ao fim de todas as práticas é necessário enviar o seu caderno de notas Jupyter na plataforma Moodle. Preencha seu caderno de notas seguindo o modelo fornecido. O nome do arquivo deve conter seu nome, sobrenome e RA.

2 Jupyter, variáveis, e como buscar ajuda

2.1 Meu primeiro caderno de notas Jupyter

Usualmente, a primeira prática quando se está aprendendo uma linguagem de programação nova é a chamada "Olá, mundo!". Nesta prática, o objetivo é criar seu primeiro programa na linguage Python que funciona sem erros.

[A] Crie um caderno de notas Jupyter seguindo o modelo discutido na Introdução. Neste caderno, crie uma célula de código contendo o código abaixo. Execute esta célula e observe o que acontece. Escreva brevemente sobre o que você observou.

```
print("Olá, mundo!")
```

Observe que no código acima temos três elementos: o comando `print`, um conjunto de parênteses, e um texto delimitado por aspas duplas. O comando `print` é o que chamamos de função. Veremos funções mais adiante, mas por hora basta saber que funções são "apelidos" para executar códigos que já foram escritos. Neste caso, `print` é uma função embutida de Python cujo código já foi escrito pelos próprios criadores do Python, por isso você pode usar esta função sem a necessidade de defini-la. A palavra *print* poderia ser traduzida para o português como *imprimir*, porém o termo *exibir* se encaixa melhor no uso corriqueiro desta função.

Para executar funções precisamos chamá-las usando os parênteses. Se tiver curiosidade, tente criar e executar uma célula no Jupyter apenas com a palavra `print` para ver o que acontece quando escrevemos o nome de uma função sem chamá-la com o parênteses.

Por fim, dentro dos parênteses que usamos para chamar a função `print` nós escrevemos os argumentos da função. Neste caso, temos apenas um argumento e ele foi o texto "Olá, mundo!".

Aprender qualquer linguagem, seja ela de programação ou não, requer treino. Não se preocupe se as regras de Python parecem esquisitas, com o passar do tempo você irá ler e escrever códigos em Python com mais facilidade. Afinal, você não aprendeu português de um dia para o outro, não é?

2.2 Criando variáveis de diferentes tipos

É comum em programação situações onde precisamos armazenar informações em variáveis para usá-las mais tarde. Python oferece diversos tipos de variáveis como, por exemplo, números inteiros (`int`), números reais (`float`), e variáveis que armazenam texto (`string`).

[A] O código abaixo cria diversas variáveis. Reescreva e execute este código em uma célula do Jupyter. Veja que para atribuir valores à variáveis nós usamos o sinal de igual (`=`). Note que para o Python, o separador decimal é o ponto (`.`) e não a vírgula (`,`); muita atenção com essa distinção pois a vírgula tem outro significado em Python e não necessariamente vai acusar um erro no seu código!! Lembrete: mesmo sendo possível copiar e colar o código, sugiro fortemente *escrever* o código abaixo pois aprendemos muito mais quando escrevemos do que quando copiamos e colamos.

Usando a função `type` e a função `print`, identifique os tipos das variáveis definidas no código que você escreveu. Exemplo: `print(type(numero_inteiro))`.

Quantos tipos diferentes você identificou? Quais foram eles?

```
texto_com_aspas_simples = 'Olá, Mundo!'
texto_com_aspas_duplas = "Olá, Mundo!"
texto_com_aspas_triplas = '''Olá, Mundo!'''

numero_inteiro = 10
numero_inteiro_positivo = +10
numero_inteiro_negativo = -10

numero_real = 10.0
numero_real_sem_digitos_depois_do_ponto = 10.
numero_real_positivo = +10.0000
numero_real_negativo = -10.0000
numero_real_base_dez = 1.5e7
pi = 3.1415

numero_complexo = 2 + 3j
numero_complexo_apenas_parte_imaginaria = 5j
numero_complexo_apenas_parte_real = 1 + 0j

variavel_booleana_verdadeiro = True
variavel_booleana_falso = False

variavel_nula = None
```

[B] Em uma célula de texto responda as perguntas:

1. Existe diferença de se criar uma string usando aspas simples, duplas, ou triplas? Reflita sobre a resposta e busque informações na internet antes de escrever. Não se esqueça de colocar as fontes na sua resposta.
2. Qual a diferença entre um número real e um número inteiro? Pelos exemplos do código acima, qual seria a regra para definir números inteiros e números reais?
3. Qual a regra para declarar números complexos? O que é o `j` na declaração dos números complexos?
4. Na sua opinião, existe diferença entre as variáveis `numero_inteiro` e `numero_inteiro_positivo`? Explique brevemente seu raciocínio.

5. Na sua opinião, existe diferença entre as variáveis `numero_real`, e `numero_real_positivo`, e `numero_real_sem_digitos_depois_do_ponto`? Explique brevemente seu raciocínio. De que forma você faria para confirmar a sua resposta?

[C] O código abaixo *supostamente* cria novas variáveis. No entanto, algumas das declarações parecem... diferente... quem sabe até estranhas! Seu objetivo é testar cada uma destas declarações e separar as que funcionam das que não funcionam. Para as declarações que não funcionam, descreva o erro que ocorreu e proponha uma correção.

```
variavel_que_o_nome_termina_com_numero_100 = 1
100_variavel_que_o_nome_comeca_com_numero = 1
numero_do_agente_secreto = 007

nome_da_variável_com_acento = True
booleano_sem_primeira_letra_maiuscula = true

espaco_entre_os_numeros = 100 000
numero_com_sublinhado = 10_000_000

numero_com_muitos_sinais_de_menos = -----10
numero_com_muitos_sinais_de_mais = +++++10
numero_com_muitos_sinais_de_mais_e_menos = +--++-10

muitos_espacos_entre_o_sinal_de_igual     =      10
nenhum_espaco_entre_o_sinal_de_igual=10

texto = "Olá, Mundo!'
```

2.3 Buscando ajuda

Nas práticas anteriores usamos a função `print` para exibir informações na forma de texto dentro do próprio caderno de notas do Jupyter. A função `print` é uma *função embutida* do Python. Isto que dizer que qualquer usuário que tenha o Python instalado (idealmente na mesma versão que a sua) terá acesso a esta função sem a necessidade de executar nenhum comando adicional.

Na prática, nós já sabemos que a função `print` recebe como argumento o texto ou variável que será exibido. Argumentos de funções são as informações

que estão dentro dos parênteses quando executamos as funções; o argumento da função `print` em `print(1234)` é o número 1234. Funções podem receber um ou mais argumentos, bem como podem receber zero argumentos (tente rodar `print()` e veja o que acontece!). Os argumentos das funções são definidos no momento quando a função é definida (vamos ver isso em mais detalhes em uma prática futura).

O que você faria se não soubesse o que função `print` faz? Digamos, por exemplo, que você viu essa fução sendo usada em um código de Python mas não sabe sua funcionalidade. Como proceder? Algumas sugestões neste caso são:

1. Pesquisar na internet em busca de alguma página com explicações (Python é uma linguagem de programação muito usada, existe muito material didático disponível online);
2. Pesquisar na documentação oficial do Python disponível em português no link `https://docs.python.org/pt-br/3/`. Todas as funções embutidas de Python, por exemplo, estão descritas aqui: `https://docs.python.org/pt-br/3/library/functions.html`;
3. Usar a função `help` do próprio Python (tente rodar `help(print)` no seu caderno de notas e veja o que acontece);
4. Usar a sintaxe própria do Jupyter para buscar ajuda. Para isso basta digitar a função que quer saber mais informações junto com um sinal de interrogação. Neste caso seria `print?`.

[A] Escolha uma das quatro sugestões acima para ler mais sobre a função `print` do Python. Observe que a função `print` aceita diferentes argumentos, sendo eles: `value`, `file`, `flush`, `end`, e `sep`. Escreva como foi sua busca (incluindo fontes se for o caso) e descreva com suas palavras o que os argumentos `end` e `sep` fazem. Tente outras formas de busca caso julge necessário.

[B] Escolha pelo menos duas funções da lista abaixo. Para cada função escolhida, faça uma busca online (buscas 1 ou 2) e uma busca offline (buscas 3 ou 4) para entender o que estas funções fazem. Escreva como foi sua busca (incluindo fontes) e descreva com suas palavras o que estas funções fazem.

- `round`
- `pow`
- `oct`
- `abs`

3 Operadores

3.1 Operadores aritméticos

A linguagem Python contém diversos operadores aritméticos como adição (`+`), subtração (`-`), multiplicação (`*`), divisão (`/`), e exponencial (`**`). Estes operadores permitem realizar cálculos aritméticos com números inteiros, reais, e complexos. Cuidado: em Python, a exponenciação é representada por dois asteriscos; não confundir com o acento circunflexo (`^`) que é o operador de exponenciação usado no Excel!

[A] Reescreva o código abaixo em uma célula do Jupyter e veja o que acontece. Comente porque o resultado da terceira linha foi `100.25` e não `5.25`.

```
print(1 + 1)
print(1 - 1 + 1 - 1)
print(-10 + 10.5 * 10.5)
print(10 / 2 * 5)
print(10 ** 10)
print(2 ** 1 / 2)
print(2 ** (1 / 2))
print((1 + 1j) / (1 - 1j))
```

A ordem de precedência dos operadores aritméticos (isto é, qual é a ordem de execução dos operadores) segue a regra do PEMDAS: parênteses, exponencial, multiplicação, divisão, adição, e subtração. Assim como na notação matemática, usamos parênteses para dar preferência para certas operações.

[B] Utilizando os operadores aritméticos do Python, compute:

1. Quantos segundos existem em 16 horas e 42 minutos?
2. Quantos centímetros existem em 72,8 milhas?
3. Se você percorer 72,8 milhas em 16 horas e 42 minutos, qual a sua velocidade média em centímetros por segundo?
4. Quanto tempo você demoraria para percorrer a circunferência da Terra na linha do equador se permanecer na velocidade média obtida no item acima?

[C] Além dos operadores discutidos acima, existem diversos outros em Python. Dois operadores bastante úteis são a divisão inteira (`//`) e o módulo

(%, também conhecido como resto da divisão inteira). Teste estes operadores e descreva como eles funcionam (em caso de dúvida, busque ajuda assim como discutido na Seção Buscando Ajuda). Discorra brevemente sobre possíveis situações onde estes operadores podem ser úteis.

Atenção: operadores aritméticos usualmente funcionam com qualquer combinação de números inteiros, reais, ou complexos. No entanto, preste atenção no tipo do resultado final!

[D] Escreva e execute o código abaixo e comente sobre o resultado obtido. Escreva um código similar ao código abaixo, porém alterando o operador de adição pelo operador de divisão. Você obteve algum resultado inusitado? Comente.

```
inteiro_mais_inteiro = 1 + 1
print(type(inteiro_mais_inteiro))

inteiro_mais_real = 1 + 1.5
print(type(inteiro_mais_real))

real_mais_real = 1.5 + 1.5
print(type(real_mais_real))

inteiro_mais_complexo = 10 + (1 - 2j)
print(type(inteiro_mais_complexo))

real_mais_complexo = 10.5 + (1 - 2j)
print(type(real_mais_complexo))
```

Desafio: durante um exercício de geometria, você decidiu realizar suas contas utilizando Python. Você escreveu o código abaixo em uma célula no seu caderno de notas Jupyter e o resultado que obteve foi inusitado! Por que o resultado é inusitado? Qual era o resultado esperado? Qual é a explicação para isso? Na sua opinião, este tipo de "problema" compromete o uso de Python como uma calculadora aritmética?

```
pi = 3.14
valor = pi + 2
print(valor)
```

3.2 Operadores lógicos

Variáveis lógicas (também conhecidas como variáveis booleanas) são objetos que podem assumir apenas dois valores diferentes: verdadeiro ou falso. Na

sintaxe de Python, escrevemos verdadeiro ou falso em inglês e com a primeira letra maiúscula: `True` ou `False`.

Uma expressão booleana é uma expressão que, quando resolvida, resulta em um valor verdadeiro (`True`) ou em um valor falso (`False`). Expressões booleanas podem ser escritas com os operadores lógicos E (`and`), OU (`or`), e NÃO (`not`). Os operadores `and` e `or` são chamados de operadores binários pois requerem sempre dois argumentos para serem resolvidos. A sintaxe para usar estes operadores com os argumentos `A` e `B`, por exemplo, é a seguinte: `A and B` e `A or B`.

[A] Escreva e execute o código abaixo; comente sobre qual ou quais situações o operador `and` retorna o valor `True`.

```
print(True and True)
print(True and False)
print(False and True)
print(False and False)
```

Se quisermos, podemos rescrever o código acima usando variáveis para facilitar a visualização:

```
A = True
B = False

print(A and A)
print(A and B)
print(B and A)
print(B and B)
```

[B] Escreva e execute um código similar ao código acima, substituindo `and` por `or`; comente sobre qual ou quais situações o operador `or` retorna o valor `True`.

O operador `not` é um operador unário; ele requer apenas um argumento para ser computado. A sintaxe para usar este operador com o argumento `A`, por exemplo, é a seguinte: `not A`.

[C] Escreva e execute o código abaixo e comente sobre como o operador `not` funciona.

```
print(not True)
print(not False)
```

Operadores lógicos podem ser combinados na mesma declaração. Sempre que for fazer uma combinação de operadores lógicos, lembre-se de usar o parênteses para garantir a ordem de execução desejada.

```
A = True
B = False
print((A or B) and not (B and A))
```

3.3 Operadores de comparação

Operadores de comparação (também conhecidos como operadores relacionais) fazem exatamente o que o nome sugere: comparam dois objetos. O resultado dessa comparação é uma variável booleana (`True` ou `False`).

Existem 6 operadores de comparação em Python:

- igualdade (`==`)
- diferença (`!=`)
- maior que (`>`)
- menor que (`<`)
- maior ou igual que (`>=`)
- menor ou igual que (`<=`)

[A] Suponha que `A = 1`, `B = 2`, `C = 4`, `D = 8`, e `E = 16`. Atribua o valor verdadeiro ou falso para cada uma das expressões abaixo. Escreva um código em Python para checar se você acertou. Comente sobre como funcionam as expressões com mais de um operador de comparação.

```
A == B
A < B
B > C
D <= E
A != E
E >= D >= C
A < C == D
A + A != B
A + A < B < C - B
A + A <= B <= C - B
A != B < C == E - D - C
```

Cuidado: é muito comum confundir o operador de atribuição (`=`) com o operador de comparação de igualdade (`==`). O operador de atribuição é usado para atribuir um valor a uma variável, por exemplo: `A = 10` atribui o valor 10 para a variável `A`. Já o operador de comparação de igualdade responde a pergunta se os objetos sendo comparados são iguais. Neste caso, a expressão `A =` 10= responde a pergunta se o valor armazenado em `A` é igual a 10, resposta esta que pode ser verdadeiro ou falso.

3.4 TODO Estrutura de decisão e operador condicional

Em diversos momentos da vida nos adaptamos mediante às condições do ambiente. Por exemplo, ao sair de casa, *se* estiver chovendo *então* pegamos o guarda-chuva, *se não* nós saímos de casa sem o guarda-chuva. Esta estrutura condicional também existe nas linguagens de programação. Em certos momentos, queremos que a execução de um determinado comando só ocorra caso uma ou mais condições sejam atendidas.

[Podemos escrever a coisa acima em python]