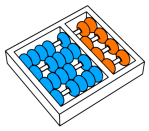

Estudo da Taxonomia de Bloom e Criação de um Instrumento de Medição do Nível de Bloom no contexto de MC102

G. Araújo *R. Caceffo* *I. Garcia* *R. Azevedo*

Technical Report - IC-20-05 - Relatório Técnico
April - 2020 - Abril

UNIVERSIDADE ESTADUAL DE CAMPINAS
INSTITUTO DE COMPUTAÇÃO

Estudo da Taxonomia de Bloom e Criação de um Instrumento de Medição do Nível de Bloom de Alunos de MC102

Guilherme Ianhes M. de Araújo* Ricardo Caceffo† Islene Garcia ‡
Rodolfo Azevedo §

Resumo

Essa pesquisa se utilizou da Taxonomia de Bloom *et al.* [1], desenvolvida com o objetivo principal de classificar o comportamento esperado de alunos diante de uma aula ou atividade, a fim de estudar possibilidades de melhorias na aprendizagem dos alunos do curso de Algoritmos e Programação de Computadores (MC102). A Taxonomia de Bloom *et al.* [1] e sua adaptação por Anderson *et al.* [2] definem uma pirâmide com 6 níveis taxonômicos que podem ser usados para classificar conteúdo, potencialmente ajudando educadores a encontrar problemas na construção de suas aulas e avaliações. Ainda, a pesquisa avaliou Concept Inventories (CIs), questionários de múltipla escolha utilizados para revelar conceitos mal entendidos (misconceptions) relacionados a um tópico ou matéria [3] [4] [5].

Os objetivos da pesquisa foram: estudar a taxonomia de Bloom *et al.* [1] e sua pirâmide adaptada [2]; estabelecer critérios de classificação de questões de MC102 de acordo com a taxonomia; usá-los para classificar as questões dos CIs já desenvolvidos por Caceffo *et al.* [5] em linguagem C e por Gama *et al.* [4] em Python e, por fim, criar um instrumento de medição do nível de Bloom *et al.* [1], aplicável a alunos de MC102.

Deste modo, inicialmente foram elaborados critérios a fim de permitir a adoção da Taxonomia de Bloom *et al.* [1] no contexto de MC102. Em seguida, os critérios foram utilizados para classificar as questões dos CI estudados. Resultados indicam que ambos os CIs [4] [5] se posicionam majoritariamente no nível 3 da taxonomia. Nenhuma questão foi classificada nos níveis 1, 4, 5 ou 6 para nenhum dos questionários. Nota-se, portanto, uma grande homogeneidade entre as questões dos CIs, o que é um possível indicativo de sua consistência.

Por fim, foi criado o Instrumento de Medição do Nível de Bloom, composto por três questões de múltipla escolha para cada um dos 27 misconceptions definidos em Python por Gama *et al.* [4], respectivamente nos três primeiros níveis da taxonomia, totalizando 81 questões.

Como trabalho futuro, pretende-se aplicar o instrumento de medição aos alunos de MC102. A hipótese de pesquisa é que, para responder uma questão de determinado nível, é necessário ter-se o conhecimento dos níveis anteriores. Assim, por exemplo, um aluno que responda corretamente uma questão de nível 3 deveria ter também acertado as questões dos níveis 1 e 2.

Palavras chave: Taxonomia de Bloom, Concept Inventory, classificação, questionário, MC102.

*Aluno de graduação do Instituto de Computação da Universidade Estadual de Campinas (UNICAMP),Campinas-SP, Brasil. g155616@dac.unicamp.br

†Pós-doutorando do Instituto de Computação da Universidade Estadual de Campinas (UNICAMP), Campinas-SP, Brasil. caceffo@ic.unicamp.br

‡Professora Doutora do Instituto de Computação da Universidade Estadual de Campinas (UNICAMP), Campinas-SP, Brasil. islene@ic.unicamp.br

§Professor Doutor do Instituto de Computação da Universidade Estadual de Campinas (UNICAMP),Campinas-SP, Brasil. rodolfo@ic.unicamp.br

1 Introdução

Essa pesquisa fez parte de um projeto maior sobre o estudo de práticas para melhorar o curso de Algoritmo e Programação de Computadores (MC102) [4–12]

Mais precisamente, realizou-se um estudo sobre a Taxonomia de Bloom [1] a fim de utilizá-la para a construção de um questionário que classifique alunos da disciplina de Algoritmos e Programação de Computadores (MC102) em níveis da taxonomia de forma a evidenciar possíveis falhas na metodologia de avaliação ou aprendizado do curso.

A Taxonomia de Bloom [1] foi desenvolvida com o objetivo principal de classificar o comportamento esperado de alunos diante de uma aula ou atividade. Segundo a taxonomia, esse comportamento esperado dos alunos está intimamente ligado aos objetivos que lhes foram estipulados durante exercícios ou explicações. Foram, então, criados níveis taxonômicos que devem ajudar educadores a encontrar problemas em suas avaliações ou na construção de suas aulas [1]. A taxonomia foi dividida em seis níveis: conhecimento (knowledge), compreensão (comprehension), aplicação (application), análise (analysis), síntese (synthesis) e avalição (evaluation). Esses níveis podem ser melhores compreendidos na Figura 1.

Figura 1: Pirâmide original da Taxonomia de Bloom [1].

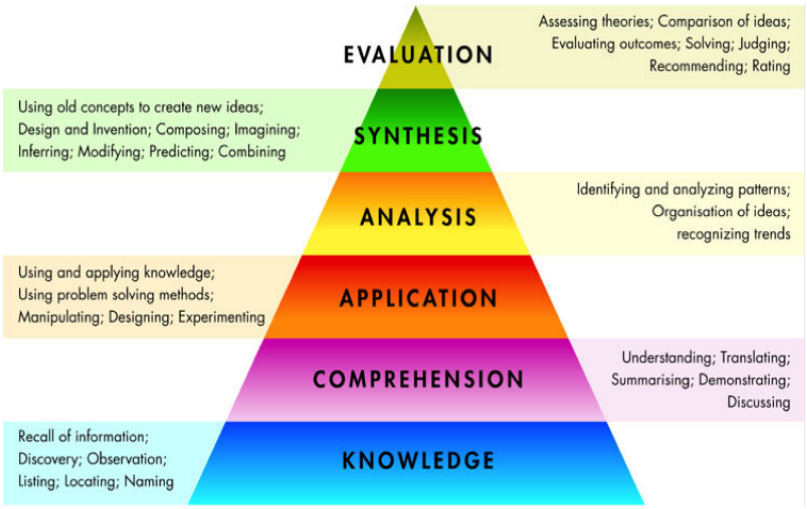

Figura 1: Fonte: Madrassah.com.uk. Disponível em https://www.madrassah.co.uk/blooms-taxonomy/. Acesso em Março de 2020.

Além da Taxonomia de Bloom [1], a pesquisa se utilizou de Concept Inventories (CIs), questionários de múltipla escolha utilizados para revelar conceitos mal entendidos (misconceptions) relacionados a um tópico ou matéria [5]. Existem dois CIs criados pelo Instituto de Computação da Universidade Estadual de Campinas e utilizados neste estudo: um deles elaborado para a linguagem C [5] e um segundo elaborado para Python [4]. Esses questionários conseguem classificar qual foi o conceito mal compreendido pelo aluno através da alternativa escolhida em cada uma das questões. Sendo assim, mostrou-se interessante avaliar quais níveis da taxonomia são melhores compreendidos por esses questionários já existentes, tanto quanto criar um novo questionário que, para cada uma das novas misconceptions definidas por Gama *et al.* [4] em Python, classifique os alunos de acordo com a Taxonomia de Bloom adaptada [2].

O relatório está organizado da seguinte forma: na **Seção 2** existe uma justificativa que motivou a

execução da pesquisa. A **Seção 3**, por sua vez, apresenta os objetivos a serem atingidos pelo estudo. A**Seção 4** expõe a metodologia do projeto, compreendendo em maiores detalhes os processos que possibilitaram atingir os objetivos. Ainda, a **Seção 5** agrupa os resultados do projeto, enquanto a **Seção 6** faz uma discussão sobre o significado dos resultados atingidos. Por último, a **Seção 7** conclui o processo de pesquisa.

Adicionalmente, o **Apêndice A** apresenta as questões elaboradas e utilizadas nesta pesquisa.

2 Justificativa

Acreditamos que a classificação dos alunos pela taxonomia de Bloom *et al.* [1] em relação à sua compreensão do conteúdo de MC102 pode ajudar educadores e alunos a identificarem possíveis carências no processo de aprendizagem.

3 Objetivos

Foram definidos os seguintes objetivos de pesquisa:

O1: Estudar a teoria da taxonomia de Bloom *et al.* [1] e sua pirâmide adaptada [2].

O2: Estabelecer critérios de classificação de questões de MC102 de acordo com a taxonomia de Bloom *et al.* [1] e sua pirâmide adaptada [2].

O3: Usar os critérios estabelecidos a fim de classificar as questões dos CIs já desenvolvidos por Caceffo *et al.* [5] em linguagem C e por Gama *et al.* [4] em Python, a fim de estudar a profundidade hierárquica desses questionários.

O4: Criar de um instrumento de medição do nível de Bloom, *et al.* [1], aplicável a alunos de MC102.

4 Metodologia

4.1 Estudo da Literatura

4.1.1 Estudo da taxonomia de Bloom *et al.* [1]

O primeiro passo do projeto foi estudar a Taxonomia de Bloom *et al.* [1] e sua adaptação [2]. Esses estudos são generalizados para qualquer projeto de fim pedagógico, mas nesta pesquisa o foco foi entender como esse conteúdo pode contribuir para a construção de cursos de programação de computadores.

4.1.2 Estudo de trabalhos relacionados

Foi realizado um estudo das publicações relacionadas a este projeto, tanto sobre os CIs [5] [4] quanto qualquer outro estudo que tenha utilizado alguma versão da Taxonomia de Bloom *et al.* [1] em atividades pedagógicas relacionadas a programação de computadores.

4.2 Criação dos critérios de Classificação

Como os os níveis da Taxonomia de Bloom *et al.* [1] são generalizados para qualquer área do conhecimento, fez-se necessário a tradução de seus niveis para critérios de classificação de questões para o conhecimento específico de MC102. Foram realizadas 9 reuniões entre o grupo de pesquisa, que contou com três estudantes de Graduação da Universidade Estadual de Campinas e o pós-doutorando orientador do grupo. Nessas reuniões, os critérios foram debatidos de acordo com a técnica de Brainstorming [13]. A cada reunião, os critérios já definidos em reuniões anteriores foram revisados, bem como novos critérios foram definidos. Parte dessas reuniões foram auxiliadas pela coordenadora do curso de MC102.

4.3 Classificação e análise das questões do CI de acordo com Bloom *et al.* [1]

Usando os critérios estabelecidos pelo grupo, os CIs em C [5] e Python [4] foram classificados entre os seis níveis da taxonomia [1]. A classificação das questões foi feita, principalmente, em função dos objetivos principais do exercício, como por exemplo: interpretar a saída de um programa funcional ou traduzir uma ideia simples para poucas linhas de código.

Feita a classificação, os resultados foram analisados através de gráficos e tabelas. Com isso foi possível determinar quais foram os níveis da Taxonomia de Bloom *et al.* [1] menos explorados pelos CIs e quais seriam as razões para essa carência.

4.4 Criação do questionário

Para cada misconception do estudo de Gama *et al.* [4], foi criada uma questão para cada um dos três primeiros níveis da Taxonomia de Bloom *et al.* [1] de forma a construir um questionário utilizando a linguagem Python. Esse questionário foi criado com a intenção de posicionar os alunos na pirâmide adaptada [2] em relação a sua compreensão do conteúdo de MC102.

5 Resultados

5.1 Estudo da Literatura

5.1.1 Estudo da taxonomia de Bloom *et al.* [1]

O nosso estudo sobre a taxonomia de Bloom *et al.* [1] nos levou a encontrar uma reestruturação dessa teoria feita por Anderson *et al.* [2], apresentada na Figura 2. Anderson *et al.* [2] reestruturaram a Taxonomia de Bloom [1] em uma nova pirâmide de níveis, utilizando novos conceitos desenvolvidos pela pedagogia. Os níveis reestruturados por Anderson *et al.* [2] utilizam verbos (lembrar, entender, aplicar, analisar, avaliar e criar) para os nomes das novas categorias. Essa nova forma explica que a classificação deve ser definida por um objetivo a ser atingido pelo aluno, ou seja, uma ação.

A Tabela 1 apresenta um resumo do principais pontos da taxonomia de Bloom *et al.* [1] e de sua reestruturação [2]. Nota-se uma clara inversão dos níveis cinco e seis por parte do segundo autor.

5.1.2 Estudo de trabalhos relacionados

Oliver *et al.* [14] utilizaram a Taxonomia de Bloom *et al.* [1] para avaliar cursos de *Programming* 1, 2 e 3 e de *Data Communications and Networking* 1, 2, e 3 , dado que *Programming* 1 (P1) seria equivalente a MC102. Oliver *et al.* [14] se utilizou da taxonomia original [1] para encontrar um

Nível	**Bloom *et al.* [1]**	**Descrição**	**Anderson *et al.* [2]**	**Verbos**
1	Conhecimento	habilidades de lembrar dos fatos	Lembrar	defina, duplique, liste, memorize, repita, diga
2	Compreensão	habilidades de entender, interpretar ou traduzir os fatos	Entender	classifique, descreva, discuta explique, identifique, localize, reconheça, relate, selecione, traduza
3	Aplicação	habilidades de usar o conhecimento em um novo contexto	Aplicar	execute, implemente, solucione, use, demonstre, interprete, opere, planeje, rascunhe
4	Análise	habilidade de reconhecer relações entre dois conhecimentos	Analisar	diferencie, organize, relacione, compare, contraste, distinga, examine, experimente, questione, teste
5	Síntese	habilidade de (re)montar partes em um novo todo	Avaliar	estime, argumente, defenda, julgue, selecione, suporte, valorize, critique, pondere
6	Avaliação	habilidade de tomar decisões justificadas pelo conhecimento (re)montado	Criar	desenhe, monte, construa, conjecture, desenvolva, formule, crie, investigue

Tabela 1: Reunião das principais características da taxonomia de Bloom *et al.* [1] e de sua reestruturação [2].

número dentro do intervalo de 1 a 6 (representando os níveis da taxonomia) a fim de determinar a dificuldade dos cursos lecionados pela *regional Australian University*. Ao fim da pesquisa foi obtido um valor de 3.9, o que colocaria o curso de P1 muito próximo do nível 4 da taxonomia (análise).

Além da pesquisa de Oliver *et al.* [14], dois CIs se mostraram interessantes de se analisar de acordo com a taxonomia de Bloom *et al.* [1]: um questionário em linguagem C com 27 questões [5] e um segundo questionário em linguagem Python com 28 questões [4]. Ambos compartilham maior parte dos misconceptions encontrados em seus respectivos estudos, salvo algumas particularidades. Os resultados da classificação dos CIs pela Taxonomia de Bloom são apresentados na Seção 5.3.

5.2 Criação dos critérios de Classificação

Após as reuniões do grupo de pesquisa, foram elaborados critérios a fim de traduzir a Taxonomia de Bloom *et al.* [1] para níveis mais específicos do conhecimento de MC102. Os critérios estabelecidos se encontram na Tabela 2.

Figura 2: Pirâmide revisada da Taxonomia [2]

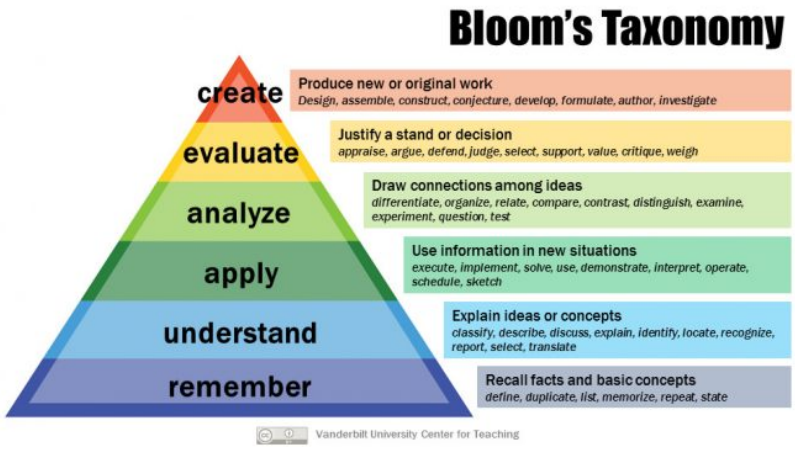

Figura 2: Fonte: Vanderbilt University Center for Teaching.

Nível	Critérios
1. Lembrar	Aquilo que é decorado pelos alunos, exercício em língua portuguesa ou em linguagem de alto nível. Deve compreender algo que intimamente associado ao que foi ensinado em sala de aula.
2. Entender	Reconhecer código, traduzir uma ideia simples para poucas linhas de código, explicar comandos ou breves trechos de código, identificar e reconhecer o uso de comandos ou ideias vistas em sala de aula ou selecionar a implementação correta de um comando visto em sala de aula.
3. Aplicar	Executar algoritmo passo a passo, construir ou corrigir trechos de código ou interpretar a saída de um trecho de código.
4. Analisar	Analisar algoritmo envolvendo complexidade, número de trocas ou comparações.
5. Avaliar	Tomar de decisões justificadas pela análise de algoritmos.
6. Criar	Criar um projeto complexo com inclusão justificada de algoritmos analisados.

Tabela 2: Critérios de classificação e criação de questões, segundo a Taxonomia de Bloom [1, 2], para a disciplina de MC102.

5.3 Classificação e análise das questões do CI de acordo com Bloom

Após a definição dos critérios para classificação das questões (Tabela 2), estes foram utilizados para classificar as questões de ambos CIs avaliados por essa pesquisa [5] [4]. Os resultados da classificação podem ser encontrados nas Tabelas 3 e 4.

Após a classificação das questões, foi feita uma análise da distribuição dos níveis de classificação para cada misconception (organizados por tópicos, nomeados de acordo com a pesquisa original [5], de A até X) em comparação com o CI completo. Esses resultados estão separados para as linguagens C e Python, representados nas Figuras 3 e 4 respectivamente.

Questão	Classificação	Critério
QA1	3	Construção ou correção de trechos de código
QA2	3	Construção ou correção de trechos de código
QA3	3	Construção ou correção de trechos de código
QA4	2	Tradução simples para poucas linhas de código
QA5	3	Interpretação da saída de um trecho de código
QB1	3	Interpretação da saída de um trecho de código
QB2	3	Interpretação da saída de um trecho de código
QB3	3	Interpretação da saída de um trecho de código
QC1	3	Construção ou correção de trechos de código
QC2	3	Construção ou correção de trechos de código
QC3	3	Construção ou correção de trechos de código
QD1	3	Interpretação da saída de um trecho de código
QD2	3	Construção ou correção de trechos de código
QD3	3	Construção ou correção de trechos de código
QD4	3	Construção ou correção de trechos de código
QD5	3	Interpretação da saída de um trecho de código
QE1	2	Tradução simples para poucas linhas de código
QE2	2	Tradução simples para poucas linhas de código
QE3	2	Tradução simples para poucas linhas de código
QE4	2	Tradução simples para poucas linhas de código
QF1	3	Construção ou correção de trechos de código
QF2	3	Interpretação da saída de um trecho de código
QF3	3	Construção ou correção de trechos de código
QF4	3	Interpretação da saída de um trecho de código
QG1	2	Tradução simples para poucas linhas de código
QG2	2	Tradução simples para poucas linhas de código
QG3	2	Tradução simples para poucas linhas de código

Tabela 3: Classificação das questões do CI em C através dos critérios definidos pela pesquisa

Questão	Classificação	Critério
QA1	3	Construção ou correção de trechos de código
QA2	3	Construção ou correção de trechos de código
QA3	3	Interpretação da saída de um trecho de código
QB1	3	Interpretação da saída de um trecho de código
QB2	3	Interpretação da saída de um trecho de código
QB3	3	Interpretação da saída de um trecho de código
QC1	3	Construção ou correção de trechos de código
QC2	3	Construção ou correção de trechos de código
QC3	3	Construção ou correção de trechos de código
QD1	3	Interpretação da saída de um trecho de código
QD2	3	Construção ou correção de trechos de código
QD3	3	Construção ou correção de trechos de código
QD4	3	Interpretação da saída de um trecho de código
QD5	3	Interpretação da saída de um trecho de código
QD6	3	Construção ou correção de trechos de código
QD7	3	Construção ou correção de trechos de código
QF1	3	Interpretação da saída de um trecho de código
QF2	3	Interpretação da saída de um trecho de código
QF3	3	Interpretação da saída de um trecho de código
QH1	3	Construção ou correção de trechos de código
QH2	3	Construção ou correção de trechos de código
QH3	3	Construção ou correção de trechos de código
QI1	3	Interpretação da saída de um trecho de código
QI2	3	Interpretação da saída de um trecho de código
QI3	3	Interpretação da saída de um trecho de código
QX1	3	Interpretação da saída de um trecho de código
QX2	3	Interpretação da saída de um trecho de código
QX3	3	Construção ou correção de trechos de código

Tabela 4: Classificação das questões do CI em Python através dos critérios definidos pela pesquisa

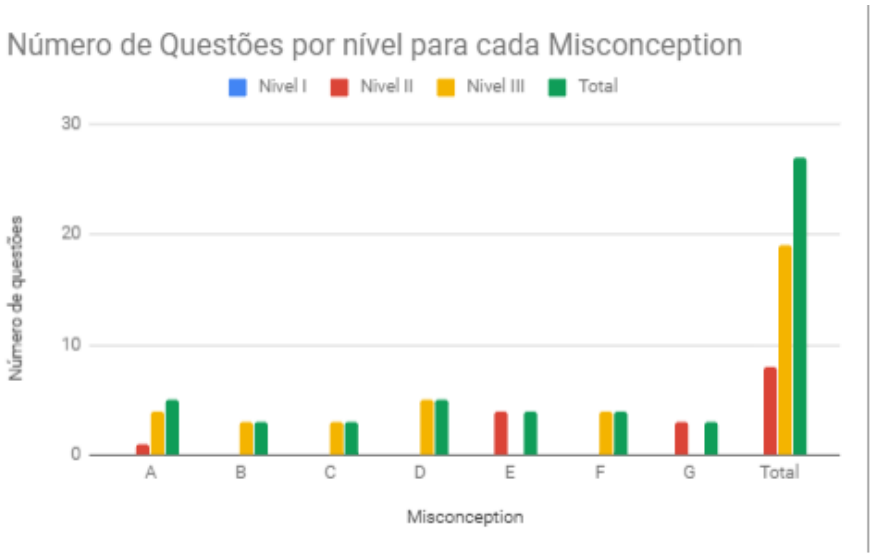

Figura 3: Número de questões por nível de classificação para cada misconception do CI em C (organizados por tópicos, nomeados de acordo com a pesquisa original [5], de A até G)

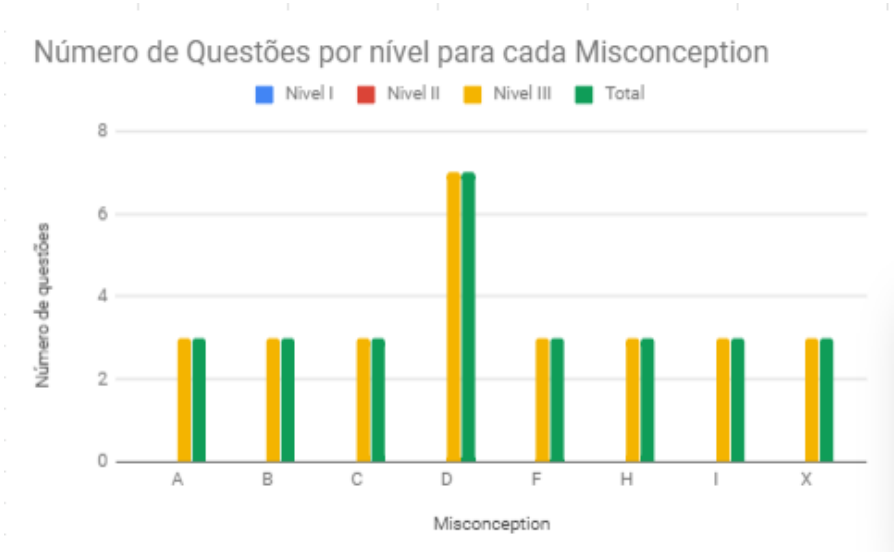

Figura 4: Número de questões por nível de classificação para cada misconception do CI em Python (organizados por tópicos, nomeados de acordo com a pesquisa original[4], de A até X)

5.4 Criação do Instrumento de Medição do Nível de Bloom

Por fim, foi elaborado o Instrumento de Medição do Nível de Bloom, composto por 81 questões de múltipla escolha (três questões para cada um dos 27 misconceptions definidos em python por Gama *et al.* [4]). Foram utilizados somente os três primeiros níveis da classificação pela taxonomia de Bloom *et al.* [1] devido a MC102 ser uma disciplina inicial aos cursos de computação, não atingindo completamente os três últimos níveis da taxonomia.

Os modelos de cada questão foram retirados dos critérios de classificação dispostos na Tabela 2. Para cada questão existe apenas uma alternativa correta. Ao contrário do CI, as alternativas incorretas não indicam necessariamente que o aluno possui um conceito mal entendido (misconception), mas sim que não compreende esse conceito para a profundidade na qual essa questão foi classificada.

Um exemplo das questões elaboradas pode ser encontrado na Tabela 5. Todas as questões podem ser encontradas no **Apêndice A**.

6 Discussão

Pode-se perceber pelas Figuras 3 e 4 que ambos os CI se posicionam no nível 3 da taxonomia de Bloom *et al.* [1]. O CI em C [5] teve 26% de duas questões classificadas no nível 2 enquanto os 74% restantes foram classificadas no nível 3. Ainda, o CI em Python [4] teve todas as suas questões posicionadas no nível 3. Nenhuma questão foi classificada nos Níveis 1, 4, 5 ou 6 para nenhum dos questionários. Nota-se, portanto, uma grande homogeneidade entre as questões dos CIs, o que reforça o fato de que esses questionários buscam conceitos mal entendidos pelos alunos, e devem fazê-lo através de questões com a mesma dificuldade. A homogeneidade do CI, por fim, é um bom indicativo de sua consistência.

As 81 questões de múltipla escolha do questionário criado neste projeto se distribuem igualmente entre os três primeiros níveis da classificação. Essa decisão foi tomada com base na fundamentalidade do curso de MC102. Por ser um curso de apenas um semestre, que proporciona o primeiro contato dos alunos de vários cursos com a programação de computadores, julgamos que o curso não tem profundidade suficiente para ultrapassar o nível 4. Essa decisão concorda com os achados do estudo de Oliver *et al.* [14], que concluiu que o curso de P1 se situa próximo do nível 4, sem ultrapassá-lo.

O questionário foi criado como um instrumento de medição do nível de Bloom, *et al.* [1], aplicável a alunos de MC102. Sendo assim, espera-se que para um determinado misconception, uma aluno que indique uma resposta correta para uma questão de nível 3, também o faça para os níveis 1 e 2 do mesmo misconception. Supondo que um aluno assinale corretamente somente as questões de nível 1 e 2 para um misconception, entende-se que para aquele conceito, o aluno consegue lembra-lo e entendê-lo, mas apresenta dificuldades em sua aplicação.

No presente momento, não foram realizadas aplicações do questionário, impossibilitando sua validação. Existem planos de que esse seja aplicado a um grupo heterogêneo de alunos, visto que a disciplina de MC102 é oferecida a alunos de diversos cursos. O formato de múltipla escolha utilizado para as questões pode ser corrigido automaticamente, possibilitando que o questionário seja aplicado a um maior número de estudantes. Ainda, o questionário poderia ser aplicado como um auxílio aos estudos, sem ligação com a aprovação do aluno, minimizando questões respondidas corretamente por acaso, o que atrapalharia a análise dos resultados.

7 Conclusão

Este estudo se utilizou da taxonomia de Bloom *et al.* [1] e de sua adaptação por Anderson *et al.* [2] para estabelecer critérios de classificação de questões de MC102. Posteriormente, esses critérios foram utilizados para classificar dois CIs previamente desenvolvidos [5] [4]. Notou-se que ambos os CIs situavam-se muito próximos ao nível 3 da classificação, o que é um indicativo de sua consistência em relação à taxonomia adotada.

Código	PB1
Nome	Tentativa de acessar variáveis locais fora do escopo.
Nível 1	Mário criou uma função fuu em seu código que atribui o valor 3 a uma variável var dessa função. Na sua função principal, Mário tenta acessar essa variável, porém ao executar seu programa recebe a mensagem: name "var" is not defined. Qual a causa dessa mensagem? a) A variável var está inserida no escopo da função fuu e não pode ser acessada por funções externas a ela. b) Para acessar o valor de var, uma nova variável de mesmo nome deve ser declarada na função principal. c) A chamada a essa função não foi feita corretamente. d) A variável var já está inserida no escopo da função principal, então nenhuma variável de mesmo nome pode ser criada em outras funções desse programa.
Nível 2	Uma função sem retorno realiza uma série de cálculos e imprime um valor na tela. Em um determinado momento, o programador percebe que necessita acessar o valor de uma variável interna a essa função em seu programa principal. Qual das seguintes alternativas apresenta uma possível solução para esse problema? a) Fazer com que essa variável seja um valor de retorno para a função e armazenar o retorno dessa função em uma variável do programa principal b) Fazer com que essa variável seja um valor de retorno para a função e usá-lo com o mesmo nome em sua função principal c) Acessar essa variável no programa principal através de uma variável de mesmo nome d) Apenas fazer com que essa variável seja um valor de retorno para a função.
Nível 3	O código abaixo representa uma tentativa de um programador aprendiz de escrever um programa que imprima o produto de dois número através do uso de uma função. Sabendo que esse código não funciona, qual das seguintes correções faria o código funcionar? Programa: 1 def fuu (a, b): 2 res = a * b 3 return res 4 5 a = float(input()) 6 b = float(input()) 7 print (res) a) Adição da linha "res = fuu(a, b)" entre as linhas 6 e 7. b) Adição da linha "fuu(a, b)" entre as linhas 6 e 7. c) Adição da linha: "res = float(input())" entre as linhas 6 e 7. d) Remoção da linha 3.

Tabela 5: Instrumento de Medição do Nível de Bloom: exemplo de questões de Nível 1 a 3 para o misconception PB1.

Por fim, foi elaborado o Instrumento de Medição do Nível de Bloom, composto por 81 questões igualmente divididas entre os três primeiros níveis da classificação, a fim de medir o nível de Bloom *et al.* [1] de acordo com os misconceptions mapeados previamente em Python [4]. Trabalhos futuros envolvem a validação deste instrumento, bem como sua adaptação para a linguagem C.

Referências

1 BLOOM, B. et al. *Taxonomy of Educational Goals*. [S.l.]: David McKay Company, 1956. ISBN 0679302093, 9780679302094.

2 ANDERSON, L. W.; R., K. D.; AIRASIAN, P. W. *A Taxonomy for Learning, Teaching, and Assessing: A Revision of Bloom's Taxonomy of Educational Objectives*. [S.l.]: Pearson, 2001. ISBN 080131903X, 9780679302094.

3 ALMSTRUM, V. L. et al. Concept inventories in computer science for the topic discrete mathematics. *SIGCSE Bull.*, ACM, New York, NY, USA, v. 38, n. 4, p. 132–145, jun. 2006. ISSN 0097-8418. Disponível em: ⟨http://doi.acm.org/10.1145/1189136.1189182⟩.

4 GAMA, G. et al. An antipattern documentation about misconceptions related to an introductory programming course in python. In: *Technical Report IC-18-19*. Campinas, SP, Brasil: Institute of Computing, University of Campinas, 2018. (IC-18-19), p. 1–106.

5 CACEFFO, R. et al. Developing a computer science concept inventory for introductory programming. In: *Proceedings of the 47th ACM Technical Symposium on Computing Science Education*. New York, NY, USA: ACM, 2016. (SIGCSE '16), p. 364–369. ISBN 978-1-4503-3685-7. Disponível em: ⟨http://doi.acm.org/10.1145/2839509.2844559⟩.

6 CACEFFO, R. et al. An exploratory questionnaire to support the identification and assessment of misconceptions in cs1 courses based on c programming language. In: *Technical Report IC-18-16*. Campinas, SP, Brasil: Institute of Computing, University of Campinas, 2018. (IC-18-16), p. 1–41.

7 SOUZA, R. et al. An antipattern documentation about possible misconceptions related to introductory programming courses (cs1) in java. In: *Technical Report IC-18-20*. Campinas, SP, Brasil: Institute of Computing, University of Campinas, 2018. (IC-18-20), p. 1–42.

8 CACEFFO, R. et al. A Concept Inventory for CS1 Introductory Programming Courses in C. In: *Technical Report IC-18-06*. Campinas, SP, Brasil: Institute of Computing, University of Campinas, 2018. (IC-18-06), p. 1–107.

9 CACEFFO, R.; GAMA, G.; AZEVEDO, R. Exploring active learning approaches to computer science classes. In: *Proceedings of the 49th ACM Technical Symposium on Computer Science Education*. New York, NY, USA: ACM, 2018. (SIGCSE '18), p. 922–927. ISBN 978-1-4503-5103-4. Disponível em: ⟨http://doi.acm.org/10.1145/3159450.3159585⟩.

10 CACEFFO, R. et al. An Antipattern Documentation about Misconceptions related to an Introductory Programming Course in C. In: *Technical Report IC-17-15*. Campinas, SP, Brasil: Institute of Computing, University of Campinas, 2017. (IC-17-15), p. 1–43.

11 CACEFFO, R. et al. Identifying and validating java misconceptions – complementary material. In: *Technical Report IC-19-05*. Campinas, SP, Brasil: Institute of Computing, University of Campinas, 2019. (IC-19-05), p. 1–49.

12 CACEFFO, R. et al. Identifying and validating java misconceptions toward a cs1 concept inventory. In: *Proceedings of the 2019 ACM Conference on Innovation and Technology in Computer Science Education*. New York, NY, USA: Association for Computing Machinery, 2019. (ITiCSE '19), p. 23–29. ISBN 9781450368957. Disponível em: ⟨https://doi.org/10.1145/3304221.3319771⟩.

13 LAZAR, J.; FENG, J. H.; HOCHHEISER, H. *Research Methods in Human-Computer Interaction*. [S.l.]: Wiley Publishing, 2010. ISBN 0470723378, 9780470723371.

14 OLIVER, D. et al. This course has a bloom rating of 3.9. In: *Proceedings of the Sixth Australasian Conference on Computing Education*. Darlinghurst, Sydney, Australia: Australian Computer Society, Inc, 2004. (ACE '04), p. 227–231.

Apêndice A

Questões em diferentes níveis de Bloom em relação às Misconceptions

Esse apêndice está organizado do seguinte modo: na seção A.1 são apresentados os critérios de criação e classificação de questões adotados, definidos a partir da Taxonomia de Bloom; na seção A.2 é apresentado um *overview*, para cada misconception, das questões criadas segundo os 3 primeiros níveis da Taxonomia de Bloom, bem como o status de seu processo de revisão e; na seção A.3 são apresentadas as questões criadas, organizadas por tópico e misconception.

A.1) Critérios de Criação e Classificação de questões

Os critérios adotados para criação e classificação de questões segundo a Taxonomia de Bloom, em seus 4 primeiros níveis, foram esses:

1. *Remember:*
    - Questão intimamente ligada com o que foi visto em sala de aula
2. *Understand:*
    - Tradução de uma ideia simples para poucas linhas de código
    - Tradução de poucas linhas de código para uma ideia simples
    - Explicação de comandos ou breves trechos de código
    - Identificar e reconhecer o uso de comandos ou ideias vistas em sala de aula
    - Selecionar a implementação correta de um comando visto em sala de aula.
3. *Apply:*
    - Construção de programa a partir de trechos de código
    - Correção de trechos de código
    - Interpretação da saída de um programa funcional
4. *Analyze:*
    - Comparação entre dois programas funcionais
    - Análise do número de repetições (ideia de complexidade)

A.2) Controle de Criação e Revisão

A Tabela A.1, a seguir, apresenta, para cada misconception, o overview da questão criada, incluindo a alternativa correta, bem como as alternativas incorretas que mapeiam o misconception associado.

Misconception	Questão do nível	Criada	Revisada	Resposta correta	Alternativa(s) mapeada(s) ao misconception
PA1	1	OK	OK	a	b, c, d
	2	OK	OK	a	b
	3	OK	OK	a	b, c

PA3	1	OK	OK	a	b, c, d
	2	OK	OK	a	b
	3	OK	OK	a	b
PA5	1	OK	OK	a	b, c, d
	2	OK	OK	a	b, c
	3	OK	OK	a	b, c ,d
PB1	1	OK	OK	a	b, c, d
	2	OK	OK	a	b, c
	3	OK	OK	a	b
PB3	1	OK	OK	a	b
	2	OK	OK	a	b
	3	OK	OK	a	b
PB5	1	OK	OK	a	b, c, d
	2	OK	OK	a	b
	3	OK	OK	a	b
PB6	1	OK	OK	a	b, c, d
	2	OK	OK	a	b, d
	3	OK	OK	a	b
PC1	1	OK	OK	a	b, c, d
	2	OK	OK	a	b, c, d
	3	OK	OK	a	b, c, d

PC2	1	OK	OK	a	b
	2	OK	OK	a	b
	3	OK	OK	a	b
PC3	1	OK	OK	a	b, c, d
	2	OK	OK	a	b, c, d
	3	OK	OK	a	b
PD1	1	OK	OK	a	b, c, d
	2	OK	OK	a	b, c, d
	3	OK	OK	a	b, c, d
PD2	1	OK	OK	a	b, c
	2	OK	OK	a	b, c
	3	OK	OK	a	b, c
PD3	1	OK	OK	a	b, c, d
	2	OK	OK	a	b, c, d
	3	OK	OK	a	b, c, d
PD4	1	OK	OK	a	b, c, d
	2	OK	OK	a	b, c, d
	3	OK	OK	a	b, c, d
PD5	1	OK	OK	a	b, c, d
	2	OK	OK	a	b, c, d
	3	OK	OK	a	b, c, d

PD6	1	OK	OK	a	b, c, d
	2	OK	OK	a	b, c, d
	3	OK	OK	a	b, c, d
PD7	1	OK	OK	a	b, c, d
	2	OK	OK	a	b, c
	3	OK	OK	a	b
PD8	1	OK	OK	a	b
	2	OK	OK	a	b
	3	OK	OK	a	b
PG1	1	OK	OK	a	b, c, d
	2	OK	OK	a	b, c, d
	3	OK	OK	a	b, c,
PG2 *Questões não foram criadas para esse misconception*					
PG4	1	OK	OK	a	b, c
	2	OK	OK	a	b, c
	3	OK	OK	a	b, c, d
PH1	1	OK	OK	a	b, c, d
	2	OK	OK	a	b, c, d
	3	OK	OK	a	b, c, d

PH2	1	OK	OK	a	b
	2	OK	OK	a	b
	3	OK	OK	a	b
PH3	1	OK	OK	a	b
	2	OK	OK	a	b
	3	OK	OK	a	b
PH4	1	OK	OK	a	b
	2	OK	OK	a	b
	3	OK	OK	a	b
PH5	1	OK	OK	a	b
	2	OK	OK	a	b, c, d
	3	OK	OK	a	b, c, d
PH6	1	OK	OK	a	b
	2	OK	OK	a	b
	3	OK	OK	a	b
PH7	1	OK	OK	a	b
	2	OK	OK	a	b
	3	OK	OK	a	B

Tabela A.1. Status da criação das questões; indicação de qual é a alternativa correta e, dentre as incorretas, quais foram mapeadas para o misconception associado à questão.

A.3) Questões

As subseções abaixo apresentam, para cada tópico e misconception, as questões criadas segundo a Taxonomia de Bloom (linhas Nível 1, Nível 2 e Nível 3) Os misconceptions apresentados têm um código que segue a nomenclatura *PX.N*, onde P indica que aquele é um misconception em Python, X a

sua categoria e N o seu número naquela categoria. Por exemplo, PA.1 é o primeiro misconception da categoria A.

A.3.1) Categoria A: uso e escopo de parâmetros de uma função.

PA.1

Código:	PA.1
Nome:	Definição do valor do parâmetro por fonte externa
Nível 1	Qual das alternativas abaixo possui uma afirmação incorreta sobre a sintaxe de declaração de funções em python? a) Os parâmetros recebidos por uma função são inicialmente vazios e devem receber seus respectivos valores no corpo da função. b) A declaração de uma função sempre se inicia com a partícula def. c) Os parâmetros de uma função devem ser declarados dentro de parênteses, separados por vírgulas, logo após o nome que a define. d) Uma função pode ou não ter um valor de retorno.

Nível 2	Qual das seguintes declarações da função *soma* calcula corretamente a soma de 2 e 3 através da chamada do programa abaixo?
	
	Programa:
	a = 2
	b = 3
	print (soma (a, b))
	
	a)
	def soma (a, b):
	  return a + b
	b)
	def soma (a, b):
	  a = int(input())
	  b = int(input())
	  return a + b
	c)
	def soma ():
	  return a + b
	d)
	def soma (a, b):
	  return a - b

Nível 3	Considere o seguinte programa:

	def fatorial(a):
	  # Linha 2
	  fat = 1
	  while a > 0:
	  fat = fat *a
	  a -= 1
	  return fat
	
	print(fatorial(4))
	
	Existe algum código a ser inserido na linha 2, de forma que o programa imprima o fatorial do número 4? Se sim, qual?
	a) Não é necessário inserir nenhum código.
	b) a = int(input())
	c) a = input()
	d) Não sei.

PA.3

Código:	PA.3
Nome:	Tentativa de acessar um parâmetro fora de seu escopo
Nível 1	O que significa escopo de uma variável? a) Escopo é uma característica que determina onde uma variável pode ser utilizada como um identificador em um programa, como por exemplo dentro de uma função. b) Escopo é a característica que determina o tipo de uma variável, como por exemplo se ela é um inteiro ou String. c) Escopo é a característica que determina um intervalo possível para valores que podem ser assumidos por uma variável, como por exemplo um intervalo entre os números 2 e 5 ou as letras ‘d’ e ‘j’ d) Escopo é a característica que determina se uma variável pode ser utilizada como uma lista ou como um dicionário.
Nível 2	Considere o seguinte trecho de código: def fuu(): var = 5 var = 3 fuu() print (a) Sobre a função fuu,, pode-se dizer que: a) A função cria uma nova variável var, a qual é atribuído o valor 5. b) A função altera o valor da variável var de 3 para 5.

	c) A função apresenta erros de sintaxe, pois não possui parâmetros. d) A função altera o valor de qualquer variável que receber como parâmetro para 5.

Nível 3	A seguir temos um programa cujo objetivo era calcular e imprimir a soma do quadrado de um número com a sua metade, porém, uma de suas linhas foi acidentalmente deletada.
	
	`def quadrado (a):`
	`  b = a*a`
	`  return b`
	
	`def metade(a):`
	`  c = a/2`
	`  return c`
	
	`a = float(input())`
	`#LINHA ACIDENTALMENTE DELETADA`
	`print(d)`
	
	Qual das linhas abaixo completa corretamente a linha deletada do programa?
	
	a) d = quadrado(a) + metade(a)
	b) d = b + c
	c) print(b + c)
	d) print(a)

PA.5

Código:	PA.5
Nome:	Retorno de uma função não é utilizado pela função que a chamou.
Nível 1	O que é o retorno de uma função? a) É um valor devolvido pela função, o qual pode ser utilizado pela função ou módulo que a chamou. b) É o ponto para o qual um programa deve retornar quando finalizada uma função. c) É o ponto para o qual um programa deve retornar quando iniciada uma função. d) É um valor escrito na tela que indica os resultados dos cálculos de uma função.

Nível 2	Considere que existam duas funções: soma1 e sub1 que recebem um número natural como parâmetro e retornam, respectivamente, o sucessor e o antecessor desse valor. Uma linha de código que calcula e armazena corretamente o produto do sucessor de 3 pelo seu antecessor, permitindo que esse valor possa ser futuramente usado pelo programa, seria: a) var = soma1(3) * sub1(3) b) soma1(3) * sub1(3) c) print(soma1(3) * sub1(3)) d) var = print(soma1(3) * sub1(3))

Nível 3	A seguir temos um programa cujo objetivo era calcular e imprimir a média aritmética de duas notas.
	
	def medArit (a, b):
	  c = (a+b)/2
	  return c
	
	a = float(input())
	b = float(input())
	#LINHA ACIDENTALMENTE DELETADA
	
	Porém, uma das linhas do programa foi acidentalmente deletada. Selecione a opção que faz com que o programa **não** funcione conforme o esperado.
	
	a) medArit(a, b)
	  print (a, b)
	
	b) print (medArit(a, b))
	
	c) resultado = medArit(a, b)
	  print (resultado)
	
	d) x = medArit(b, a)
	  print (x)

Categoria B: Variáveis, identificadores e escopo.

PB.1

Código:	PB.1
Nome:	Tentativa de acessar variáveis locais fora do escopo.
Nível 1	Mário criou uma função *fuu* em seu código que atribui o valor 3 a uma variável var dessa função. Na sua função principal, Mário tenta acessar essa variável,

<table>
<tr><td></td><td>porém ao executar seu programa recebe a mensagem: name “var” is not defined. Qual a causa dessa mensagem?
a) A variável var está inserida no escopo da função fuu e não pode ser acessada por funções externas a ela.
b) Para acessar o valor de var, uma nova variável de mesmo nome deve ser declarada na função principal.
c) A chamada a essa função não foi feita corretamente.
d) A variável var já está inserida no escopo da função principal, então nenhuma variável de mesmo nome pode ser criada em outras funções desse programa.</td></tr>
<tr><td>Nível 2</td><td>Uma função sem retorno realiza uma série de cálculos e imprime um valor na tela. Em um determinado momento, o programador percebe que necessita acessar o valor de uma variável interna a essa função em seu programa principal. Qual das seguintes alternativas apresenta uma possível solução para esse problema?
a) Fazer com que essa variável seja um valor de retorno para a função e armazenar o retorno dessa função em uma variável do programa principal
b) Fazer com que essa variável seja um valor de retorno para a função e usá-lo com o mesmo nome em sua função principal
c) Acessar essa variável no programa principal através de uma variável de mesmo nome
d) Apenas fazer com que essa variável seja um valor de retorno para a função.</td></tr>
<tr><td>Nível 3</td><td>O código abaixo representa uma tentativa de um programador aprendiz de escrever um programa que imprima o produto de dois número através do uso de uma função. Sabendo que esse código não funciona, qual das seguintes correções faria o código funcionar?
Programa:

1 def fuu (a, b):
2 res = a * b
3 return res
4
5 a = float(input())
6 b = float(input())
7 print (res)

a) Adição da linha “res = fuu(a, b)” entre as linhas 6 e 7.
b) Adição da linha “fuu(a, b)” entre as linhas 6 e 7.
c) Adição da linha: “res = float(input())” entre as linhas 6 e 7.
d) Remoção da linha 3.</td></tr>
</table>

PB.3

Código:	PB.3

Nome:	Confusão de parâmetro com variável de mesmo nome fora da função
Nível 1	Quando o parâmetro de uma função tem o mesmo nome de uma variável de fora desta: a) O nome fará referência ao parâmetro b) O nome fará referência a variável externa c) Python decidirá qual variável usar de acordo com os tipos utilizados d) O programa apresentará erros na hora da execução
Nível 2	No trecho de código abaixo, o que será impresso? a = 10 def fuu(a): print(a) fuu(5) a) O número 5 b) O número 10 c) Um erro pois o parâmetro da função tem o mesmo nome de uma variável externa. d) A letra a

Nível 3	Gabriela escreveu uma função com a intenção verificar se um número pertence a uma determinada lista. Em seguida escreveu um pequeno programa para testar essa função:
	
	def procura(lista, n):
	for elem in lista:
	if elem == n:
	return True
	return False
	
	lista = [1, 4, 8]
	n = 4
	resultado = procura(lista, 0)
	print(resultado)
	
	O programa de Gabriela terá como saída:
	b) False, pois o número zero não será encontrado na lista.
	a) True, pois o número quatro será encontrado na lista
	c) False, pois não há um comando condicional antes de retornar o valor False.
	d) Um erro pois o parâmetro da função tem o mesmo nome de uma variável externa.

PB.5

Código:	PB.5
Nome:	Variável de iteração do for considerada local.
Nível 1	Considere as seguintes afirmações sobre o comando condicional de repetição *for*. Assinale a afirmação **falsa.** a) O comando for tem seu próprio escopo, ou seja, as variáveis criadas internamente ao bloco do comando não podem ser acessadas em outras partes do programa. a) O comando *for* contém uma variável iteradora que receberá valores pertencentes a uma certa lista. b) O comando for executa um bloco de código repetidas vezes até que acabem os elementos em uma determinada lista. c) A variável iteradora pode ser uma das variáveis definidas previamente no escopo do programa ou uma variável nesse mesmo escopo que será definida pelo próprio comando.
Nível 2	Considere o pequeno trecho de código: i ='casa' lista = [1, 2, 3, 4] Indique, se possível, qual das seguintes linhas de código escreve corretamente a estrutura de um comando de repetição *for* que faça com que a variável i receba sequencialmente os valores da lista. a) for i in lista: c) Isto não é possível, visto que qualquer variável inserida como iteradora fará parte de um novo escopo. d) Isto não é possível, visto que a variável i já foi definida anteriormente. b) for i in range(lista):
Nível 3	O que o programa a seguir irá imprimir? i = 10 a = 20 lista = [2, 4, 8] for i in lista: a += i print ("a = ", a, "i = ", i) a) a = 34 i = 8 c) a = 34 i = 10 b) a = 34 i = [2, 4, 8] d) a = 20 i = 10

PB.6

Código:	PB.6
Nome:	Retorno de uma função muda automaticamente quando uma das variáveis previamente passadas como parâmetro muda de valor.
Nível 1	Dada uma função que calcula o quadrado de um número, Pedro quer imprimir na tela de seu computador o quadrado de um número tanto quanto o quadrado do seu sucessor. Pedro seguiu o seguinte algoritmo para cumprir sua tarefa: 1) Armazenou o valor desejado em uma variável num. 2) Chamou a função que calcula quadrados passando num como parâmetro e armazenou o seu retorno em uma variável x. 3) Imprimiu x. 4) Incrementou a variável num em uma unidade. 5) Imprimiu x. O algoritmo implementado por Pedro não funciona para cumprir o seu objetivo. Por que? a) Pedro não chama a função que calcula quadrados uma segunda vez para atualizar o valor da variável x. b) Pedro não pode usar a mesma variável para armazenar dois retornos diferentes de uma função c) Pedro incrementou no lugar de decrementar a variável num em uma unidade. d) Pedro Chama a função que calcula quadrados antes incrementar o valor da variável num.

Nível 2	Dada uma função *area(lado)* que retorna a área de quadrados, qual dos seguintes trechos de código imprimiria corretamente a área de dois quadrados de lados l1 e l2?
	
	b)
	`print (area (l1))`
	`print (area (l2))`
	
	a)
	`res = area(l1)`
	`print (res)`
	`l1 = l2`
	`print (res)`
	
	c)
	`a = l1`
	`res = area(a)`
	`print (res)`

	a = l2 print (res) d) res = area(l1) print (res) res = l2 print (res)

Nível 3	Qual será a saída do programa abaixo?
	
	def pertence (lista, elemento):
	if elemento in lista:
	return True
	else:
	return False
	
	l = [10, 20, 30]
	a = 10
	existe_na_lista = pertence(l, a)
	if existe_na_lista:
	print(a, "está na lista")
	a = 15
	if existe_na_lista:
	print(a, "está na lista")
	a = 30
	if existe_na_lista:
	print(a, "está na lista")
	
	a)
	10 está na lista
	15 está na lista
	30 está na lista
	b)
	10 está na lista
	30 está na lista
	c)
	10 está na lista
	d)
	15 está na lista
	25 está na lista

Categoria C: Recursão.

PC.1

Código:	PC.1
Nome:	Uso de fórmula errada para calcular o valor de retorno de uma função recursiva.
Nível 1	Sabemos que uma função **recursiva** resolve um problema parcialmente, chamando a si mesma para resolver o restante. Queremos fazer uma função recursiva que calcula a soma dos elementos de uma lista. Para isso, essa função recebe em sua chamada uma lista de números. Quando essa função recebe uma lista vazia, retorna zero. Caso contrário, o retorno dessa função **recursiva** deve ser: a) uma soma entre o primeiro elemento da lista e a chamada a si mesma passando como argumento uma lista com o segundo elemento em diante. b) uma chamada a si mesma passando como argumento uma lista com o segundo elemento em diante. c) uma soma entre o primeiro elemento da lista com o retorno da chamada a si mesma passando como argumento a lista recebida nesta chamada. d) uma chamada a si mesma passando como argumento a lista recebida nesta chamada.
Nível 2	Queremos construir uma função recursiva que calcula o fatorial de um número através da seguinte assinatura: `def fatorial(n):` Sabemos que o caso base dessa função checa se n é zero, retornando 1 nesse caso. Qual deve ser o retorno da função caso contrário? Caso necessário, pesquise a definição matemática do fatorial de um número! a) `return n*fatorial(n-1)` b) `return n*fatorial(n+1)` c) `return fatorial(n-1)` d) `return fatorial(n+1)`

Nível 3	A função **recursiva** fuu a seguir, recebe uma lista com um número par de elementos e retorna uma lista com metade do número de elementos tal que o primeiro elemento da nova lista é o resultado da subtração entre o primeiro e segundo elementos da lista original, o segundo elemento da nova lista é o resultado da subtração entre o terceiro e quarto elementos da lista original, e assim por diante.
	
	`def fuu(l):`
	`  if len(l) == 2:`

	return [l[0]-l[1]] else: primeiro = l[0]-l[1] ''' RETORNO DA FUNÇÃO RECURSIVA ''' Qual das alternativas indica um retorno correto para a função recursiva fuu? a) return [primeiro] + fuu(l[2:]) b) return fuu([primeiro] + l[2:]) c) return [primeiro] + fuu(l) d) return fuu(l[2:])

PC.2

Código:	PC.2
Nome:	Ausência de chamada recursiva
Nível 1	Sabemos que uma função **recursiva** resolve um problema parcialmente, chamando a si mesma para resolver o restante. Todas as alternativas abaixo explicam partes **indispensáveis** de uma função recursiva, **exceto:** a) Retorno: um valor que deve ser retornado a chamada anterior, garantindo assim a continuidade da recursão. b) Chamada recursiva: parte da função que chama a si mesma, criando assim a recursão. c) Caso base: parte da função recursiva que checa se a sua entrada deve parar de recorrer. d) Convergência: as entradas de cada recursão devem sempre se aproximar de uma entrada para a qual nenhuma chamada recursiva será feita.
Nível 2	Queremos montar uma função **recursiva** "fuu(l)", que calcula a soma dos elementos de uma lista. Existem muitas formas de se calcular o retorno dessa função que não pertence ao caso base. Uma delas **não** pode ser: a) return l[len(l) - 1] + l[:len(l)-1] b) return l[0] + fuu(l[1:]) c) return l[len(l) - 1] + fuu(l[:len(l)-1]) d) return l[1] + fuu([l[0]] + l[2:])

Nível 3	Observe a seguinte função **recursiva** que encontra o maior elemento de uma lista:
	
	def fuu(l):
	if len(l) == 1:

	return l[0]
	else:
	primeiro_elemento = l[0]
	
	if maior_resto > primeiro_elemento:
	return maior_resto
	return primeiro_elemento
	Qual das seguintes alternativas completa a linha acidentalmente deletada?
	a) maior_resto = fuu(l[1:])
	b) maior_resto = l[1:]
	c) Não sei.

PC.3

Código:	PC.3
Nome:	Função não termina no caso base.
Nível 1	Para criarmos uma função **recursiva** que soma os elementos de uma lista devemos ter um caso base. Sabendo que essa função pode receber listas vazias, qual das alternativas poderia ser um caso base? a) Quando a lista estiver vazia, retorne o valor zero. b) Quando a lista tiver apenas um elemento retorne este elemento. c) Quando a lista tiver dois elementos retorne sua soma. d) Quando a lista tiver apenas três elementos, retorne a soma desses três elementos.

Nível 2	Qual das seguintes funções **recursivas** calcula corretamente x^y?
	
	a)
	def pot(x, y):
	if y == 1:
	return x
	return x * pot(x, y-1)
	
	b)
	def pot(x, y):
	if y > 0:
	return x
	return x * pot(x, y-1)
	
	c)
	def pot(x, y):

	return x * pot(x, y-1)
	
	d)
	def pot(x, y):
	if y > x:
	return x
	return x * pot(x, y-1)

Nível 3	Considere a seguinte função recursiva que decompõe um número em fatores primos:
	
	def dec(n, fat):
	if n % fat == 0:
	print(fat)
	dec(n/fat, fat)
	else:
	dec(n, fat+1)
	
	A função funciona?
	a) Não, pois a função gerará um loop infinito.
	b) Não, pois não existe um valor de retorno.
	c) Sim.
	d) Não sei.

Categoria D: Iteração.

PD.1

Código:	PD.1
Nome:	Atualização imprópria do contador de um loop.
Nível 1	Quando um programador se utiliza do comando *while* de repetição para repetir um bloco de código dez vezes, deve garantir que: a) exista um contador (inicialmente 1) incrementado a cada iteração e que a condição de repetição cheque se esse contador ultrapassou o número dez. b) existe um contador (inicialmente 0) que receba o valor dez a cada iteração. c) existe um contador (inicialmente 10) incrementado a cada iteração e que a condição de repetição cheque se esse contador ultrapassou o número dez. d) existe um contador (inicialmente 10) que receba o valor 1 em toda iteração.
Nível 2	Um exemplo correto do uso do comando *while* para repetir um certo número de vezes um bloco de código é: a)

	i = 5
	while i > 0:
	''' BLOCO DE CÓDIGO '''
	i -= 1
	b)
	i = 0
	while i > 0:
	''' BLOCO DE CÓDIGO '''
	i -= 1
	c)
	i = 10
	while i > 0:
	''' BLOCO DE CÓDIGO '''
	i += 1
	d)
	i = 0
	while i < 5:
	''' BLOCO DE CÓDIGO '''
	i -= 1

Nível 3	Leia atentamente o programa abaixo que imprime na tela todos os elementos de uma lista. Em seguida indique qual alternativa o completa corretamente. lista = ["a", "b", "c", "d", "e"] n = len(lista) -1 while (n >= 0): print(lista[n]) '''INSIRA UMA LINHA AQUI''' a) n -= 1 b) n += 1 c) n = 1 d) n -= n

PD.2

Código:	PD.2
Nome:	Uso do resultado de um loop antes que este termine.
Nível 1	Uma estudante de programação escreveu um trecho de código para contar quantos elementos de uma lista são pares usando o comando *for*. Sendo assim,

<table>
<tr><td></td><td>para cada elemento da lista, sempre que o resto da divisão inteira de um elemento por dois resultasse em zero, um contador é incrementado em uma unidade. Agora a estudante deseja imprimir esse resultado na tela usando a função print(). Isso deve ser feito:
a) Logo abaixo das linhas internas ao bloco do comando de repetição for.
b) Interno ao bloco do comando de repetição for, na última linha do bloco.
c) Interno ao bloco do comando de repetição for, na primeira linha do bloco.
d) Antes de iniciar o bloco do comando de repetição for.</td></tr>
<tr><td>Nível 2</td><td>Qual dos trechos de código abaixo imprime corretamente a soma dos números de 1 a 10?
a)
i = 1
soma = 0
while i <= 10:
  soma += i
  i += 1
print(soma)
b)
i = 1
soma = 0
while i <= 10:
  print(soma)
  soma += i
  i += 1
c)
i = 1
soma = 0
while i <= 10:
  soma += i
  i += 1
  print(soma)
d)
i = 1
soma = 0
print(soma)
while i <= 10:
  soma += i
  i += 1</td></tr>
<tr><td>Nível 3</td><td>Abaixo se encontra o código de um programa que, dado duas listas, soma os primeiros elementos correspondentes de cada uma, em seguida soma os segundos</td></tr>
</table>

	elementos, e assim por diante até que as listas terminem. Por fim, imprime uma lista com os resultados das somas. Qual alternativa completa corretamente esse programa?
	
	lista1 = [1, 2, 3, 4, 5]
	lista2 = [5, 3, 3, 2, 1]
	n = len(lista1)-1
	''' COMPLETE O PROGRAMA '''
	a)
	while (n >= 0):
	  lista1[n] = lista1[n] + lista2[n]
	  n -= 1
	 print(lista1)
	
	b)
	while (n >= 0):
	  lista1[n] = lista1[n] + lista2[n]
	  n -= 1
	  print(lista1)
	
	c)
	while (n >= 0):
	  print(lista1)
	  lista1[n] = lista1[n] + lista2[n]
	  n -= 1
	
	d)
	print(lista1)
	while (n >= 0):
	  lista1[n] = lista1[n] + lista2[n]
	  n -= 1

PD.3

Código:	PD.3
Nome:	Loop itera o número correto de vezes mas com o intervalo errado
Nível 1	O comando de repetição *for* muitas vezes vem acompanhado da função range(), a fim de repetir um certo bloco de código para um determinado intervalo de valores. Para que esse intervalo seja o conjunto C, tal que C = {5, 6, 7 ,8} deve-se usar:

	a) range(5, 9) b) range(4, 9) c) range(5, 8) d) range(4, 8)

Nível 2	Qual dos trechos de código abaixo imprime corretamente a soma dos números de 1 a 10?
	a)
	soma = 0
	for i in range(1, 11):
	  soma += i
	print(soma)
	b)
	soma = 0
	for i in range(0, 10):
	  soma += i
	print(soma)
	c)
	soma = 0
	for i in range(1, 10):
	  soma += i
	print(soma)
	d)
	soma = 0
	for i in range(1, 10):
	  soma += i
	print(soma)

Nível 3	O programa abaixo soma todos os elementos com índice par de uma lista. Qual das linhas de código completa o programa corretamente? Dica: zero é o primeiro índice de uma lista em python. Dica 2: zero é par. lista = [4, 6, 7, 123, 3] soma = 0 ''' ESCREVA AQUI A LINHA DO COMANDO FOR ''' soma += lista[i] print(soma) a) for i in range(0, len(lista) , 2): b) for i in range(1, len(lista) -1, 2): c) for i in range(0, len(lista) +1 , 2): d) for i in range(-1, len(lista) , 2):

PD.4

Código:	PD.4
Nome:	Controle de fluxo inadequado em um loop.
Nível 1	Um estudante de programação deseja repetir um um bloco de código cinco vezes se utilizando do comando *while*. Sabendo que sua variável iteradora "i" começa em zero e que ao final do bloco é sempre incrementada em uma unidade, qual das alternativas apresenta uma possível condição a ser checada pelo comando *while*? a) i < 5 b) i <= 5 c) i < 4 d) i < 6
Nível 2	Qual das linhas abaixo inicia corretamente um bloco para um comando de repetição *for* a fim de que aconteçam quatro iterações? a) for i in range(4, 0, -1): b) for i in range(1, 4, -1): c) for i in range(4, 0, 1): d) for i in range(1, 4, 1):

Nível 3	A fim de escrever um programa que calcule a soma de todos os números inteiros entre zero e um valor recebido como entrada, um estudante de programação escreveu o código abaixo. Sobre esse código pode-se dizer que:
	
	`n = int(input())`
	
	`if n >= 0:`
	`  soma = 0`
	`  i = 1`
	`  while i <= n:`
	`  soma += i`
	`  i += 1`
	`  print(soma)`
	`elif n < 0:`
	`  soma = 0`
	`  i = -1`
	`  while i <= n:`
	`  soma += i`
	`  i -= 1`
	`  print(soma)`

	a) Esse programa não funciona pois no caso em que o número digitado for negativo, não haverá nenhuma iteração para qualquer valor menor que -1. b) O programa funciona corretamente c) Esse programa não funciona pois no caso em que o número digitado for positivo, não haverá nenhuma iteração para qualquer valor maior que 1. d) Esse programa não funciona pois no caso em que o número digitado for positivo, acontecerão infinitas iterações.

PD.5

Código:	PD.5
Nome:	Ausência de loop, apenas uma simples iteração.
Nível 1	Quando programamos, muitas vezes precisamos repetir uma parte do nosso código um certo número de vezes. Esse número pode ou não ser conhecido a priori pelo programador. Para isso fazemos uso do(s) comando(s): a) while e for b) def c) print e input d) format

Nível 2	Qual das alternativas abaixo apresenta um trecho de código que imprime todas as palavras de uma lista que não forem a palavra “cachorro”?
	a)
	`for a in lista:`
	` if a != "cachorro":`
	`  print (a)`
	b)
	`if lista != "cachorro":`
	` print (lista)`
	c)
	`a = lista[0]`
	`if a != "cachorro":`
	` print (a)`
	
	d)
	`if lista[0] != "cachorro":`
	` print (lista)`

Nível 3	Um programa receberá uma quantidade variável de números inteiros e deverá imprimir “PAR” sempre que um número recebido for par. O último número

	recebido sempre será zero, que aparecerá somente uma vez. Indique qual dos programas abaixo cumpre corretamente esta tarefa.
	a)
	n = int(input())
	while n != 0:
	 if n % 2 == 0:
	  print("PAR")
	 n = int(input())
	b)
	n = int(input())
	if n != 0:
	 if n % 2 == 0:
	  print("PAR")
	 n = int(input())
	c)
	n = int(input())
	if n % 2 == 0:
	  print("PAR")
	d)
	n = int(input())
	if n != 0:
	 if n % 2 == 0:
	  print("PAR")

PD.6

Código:	PD.6
Nome:	Construção do loop não considera a lógica do programa e suas conexões com outras partes do código.
Nível 1	Um aluno de programação se dispõe de uma função isprime() que retorna True se o número passado como parâmetro é primo e False caso contrário. Através dela, deseja criar uma segunda função que encontra o primeiro primo de uma lista de inteiros. Sendo assim, a lógica a ser implementada pode ser: a) enquanto não chegar ao fim da lista ou nenhum número primo for encontrado, chame a função isprime() para o próximo elemento. b) enquanto não chegar ao fim da lista, chame a função isprime() para o próximo elemento. c) enquanto nenhum número primo for encontrado, chame a função isprime() para o próximo elemento.

<table>
<tr><td></td><td>d) enquanto não chegar ao fim da lista ou qualquer número primo for encontrado, chame a função isprime() para o próximo elemento.</td></tr>
<tr><td>Nível 2</td><td>Uma empresa quer construir um programa que receberá como entrada o nome de um número variável de clientes e em seguida vários números inteiros sendo esses seus respectivos telefones. um exemplo de entrada pode ser lido a seguir:

Clara
Márcio
Rodrigo
Maria
915262258
963258741
975684123
966885522

Qual das alternativas apresenta um trecho de código que insere somente o nome dos clientes em uma lista?

a)
lista_nomes = []
nome = input()
while not nome.isdigit():
 lista_nomes.append(nome)
 nome = input()

b)
lista_nomes = []
nome = input()
while nome.isdigit():
 lista_nomes.append(nome)
 nome = input()

c)
lista_nomes = []
nome = input()
while not input().isdigit():
 lista_nomes.append(nome)
 nome = input()

d)
lista_nomes = []
nome = input()
while input.isdigit():</td></tr>
</table>

	lista_nomes.append(nome) nome = input()

Nível 3	Um estudante de programação escreveu o seguinte programa a fim de calcular quantas vezes um número inteiro *a* recebido como entrada pode ser dividido pelo número *b* também recebido como entrada de forma que o resto da divisão seja zero. A seguir estão alguns exemplos:
	
	a = 8, b = 2 -> divisão pode ser feita 3 vezes.
	a = 18, b = 3 -> divisão pode ser feita 2 vezes.
	a = 13, b = 5 -> divisão pode ser feita 0 vezes.
	
	Programa:
	
	a = int(input())
	b = int(input())
	n = 0
	divisivel = True
	
	''' COMANDO WHILE '''
	if a%b == 0:
	  a = a/b
	  n += 1
	else:
	  divisivel = False
	
	print("divisão pode ser feita", n ,"vezes.")
	
	Qual das alternativas a seguir implementa corretamente o comando de repetição *while* de forma que o programa funcione corretamente?
	a) while a >= b and divisivel == True:
	b) while a >= b and divisivel == False:
	c) while divisivel == False:
	d) while a >= b:

PD.7

Código:	PD.7
Nome:	Um loop do comando for que itera para elementos de uma lista é tratado como se iterasse pelos índices.
Nível 1	Quando usamos o comando *for* de repetição para iterar em uma lista, ou seja, **desacompanhado** da função *range()*, a variável iteradora: a) assumirá os valores dos elementos dessa lista.

	b) assumirá os valores para os índices dessa lista. c) não existe. d) será inconsistente pois essa não é uma declaração válida do comando *for*.
Nível 2	Dada a lista: l = [7, 8, 9] e a declaração do comando for: for i in l: A variável iteradora *i* assumirá os valores: a) i = 7, i = 8, i = 9. b) i = 0, i = 1, i = 2. c) i = 1, i = 2, i = 3. d) i = 1, i = 2, i = 3, i = 4.

Nível 3	O programa abaixo determina se todas as palavras de uma lista começam com a mesma letra. Qual das alternativas traz a condição correta a ser checada na linha marcada?
	
	l = ["rato", "roeu", "roupa", "rei", "roma"]
	primeira_letra = l[0][0]
	flag = True
	for i in l:
	if "'CONDIÇÃO"':
	flag = False
	
	if (flag == True):
	print("Todas as palavras começam com a mesma letra!")
	else:
	print("Nem todas as palavras começam com a mesma letra...")
	
	a) primeira_letra != i[0]
	b) primeira_letra != l[i][0]
	c) primeira_letra != i
	d) primeira_letra != l[0]

PD.8

Código:	PD.8
Nome:	Um loop do comando for que itera pelos índices de uma lista é tratado como se iterasse pelos elementos.
Nível 1	Quando usamos o comando *for* de repetição para iterar em uma lista **acompanhado** da função *range()*, a variável iteradora: a) assumirá os valores para os índices dessa lista.

	b) assumirá os valores dos elementos dessa lista. c) não existe. d) será inconsistente pois essa não é uma declaração válida do comando *for*.
Nível 2	Dada a lista: l = [7, 8, 9] e a declaração do comando for: for i in range(len(l)): A variável iteradora *i* assumirá os valores: a) i = 0, i = 1, i = 2. b) i = 7, i = 8, i = 9. c) i = 3, i = 4, i = 5. d) i = 0, i = 0, i = 0, i = 0.

Nível 3	O programa abaixo determina se todas as palavras de uma lista começam com a mesma letra. Qual das alternativas traz a condição correta a ser checada na linha marcada?
	
	l = ["rato", "roeu", "roupa", "rei", "roma"]
	primeira_letra = l[0][0]
	flag = True
	for i in range(len(l)):
	if "'CONDIÇÃO'":
	flag = False
	
	if (flag == True):
	print("Todas as palavras começam com a mesma letra!")
	else:
	print("Nem todas as palavras começam com a mesma letra...")
	
	a) primeira_letra != l[i][0]
	b) primeira_letra != i[0]
	
	c) primeira_letra != i
	d) primeira_letra != l[0]

Categoria G: Expressões Booleanas.

PG.1

Código:	PG.1
Nome:	Precedência incorreta de operadores booleanos
Nível 1	Seguindo a lógica de precedência de operadores booleanos em python, deseja-se

	montar a seguinte condição: um número par pode se dizer perfeito se ele é maior que dez. Além disso, todos os números ímpares são perfeitos. Se quisermos checar se um número é perfeito, deve-se checar se o número é: a) ímpar, ou maior do que dez e par. b) ímpar ou maior do que dez, e par. c) par, ou maior do que dez e ímpar. d) par, ou ímpar e maior do que dez.
Nível 2	Enumerando-se os meses a partir do número um (isto é, janeiro = 1, fevereiro = 2, …, dez = 12), um mês tem 31 dias se seu número correspondente for ímpar e menor ou igual a 7. Também terá 31 dias se for par e maior ou igual a 8. Qual das seguintes alternativas apresenta um teste em python que implementa corretamente essas condições? a) `if m <= 7 and m % 2 == 1 or m >= 8 and m % 2 == 0:` b) `if m <= 7 and (m % 2 == 1 or m >= 8) and m % 2 == 0:` c) `if (m <= 7 and m % 2 == 1 or m >= 8) and m % 2 == 0:` d) `if m <= 7 and (m % 2 == 1 or m >= 8 and m % 2 == 0):`

Nível 3	O pagamento da fatura online de uma loja de roupas só pode ser parcelado caso o cliente possua o cartão da loja e valor da compra seja superior a 100 reais. Caso o valor da compra seja inferior a 100 reais, o cliente que possuir o cartão poderá parcelar a compra somente se o saldo em sua conta for superior ao valor da compra. O código abaixo verifica se um cliente poderá ou não parcelar a sua compra nessa loja:
	
	`''' existência do cartão (1 = sim /0 = não) '''`
	`cartao = int(input())`
	`''' valor da compra do cliente '''`
	`valor = float(input())`
	`''' saldo em conta do cliente '''`
	`saldo = float(input())`
	
	`if cartao == 1 and valor >= 100 or saldo > valor:`
	`print("parcelamento aceito")`
	`else:`
	`print("parcelamento bloqueado")`
	
	A precedência das operações booleanas no código acima não funciona para o caso proposto. Para arrumá-la, as operações devem ser reagrupadas conforme a alternativa:
	a) `if cartao == 1 and (valor >= 100 or saldo > valor):`
	b) `if (cartao == 1 and valor >= 100) or saldo > valor:`
	c) `if (cartao == 1 and saldo > valor) or valor >= 100:`
	d) `if (cartao == 1) and valor >= 100 or saldo > valor:`

PG.2

Código:	PG.2
Nome:	Encadeamento de comandos *if* quando apenas uma expressão booleana poderia ter sido usada.
Nível 1	*Não definido*
Nível 2	*Não definido*
Nível 3	*Não definido*

PG.4

Código:	PG.4
Nome:	Tentativa de avaliar uma expressão booleana através de comandos de repetição.
Nível 1	Quando programamos, em alguns momentos se faz necessário desviar o fluxo do programa, por exemplo, quando precisamos fazer algo com um número par, porém algo diferente com um número ímpar. Nesses casos utilizamos o comando: a) if b) while c) for d) print

Nível 2	Dado um número, se quisermos imprimí-lo na tela somente se for par, faremos:
	a)
	`if num % 2 == 0:`
	`print(num)`
	b)
	`while num % 2 == 0:`
	`print(num)`
	c)
	`for num % 2 == 0:`
	`print(num)`
	d)
	`print(num % 2 == 0)`

Nível 3	O programa abaixo, que teve uma linha deletada, verifica se todos os elementos

	de uma lista são dígitos.
	
	l = ["33", "22", "11", "gato", "123"]
	
	todos_digitos = True
	
	for i in l:
	  if not i.isdigit():
	  todos_digitos = False
	
	''' Linha deletada '''
	  print("são todos dígitos")
	
	Qual das linhas abaixo completa corretamente o programa?
	a) if todos_digitos == True:
	b) while todos_digitos == True:
	c) for todos_digitos == True:
	d) for i in todos_digitos == True:

Categoria H: Uso e Implementação de Classes e Objetos.

PH.1

Código:	PH.1
Nome:	Tentativa de usar um método fora de sua classe.
Nível 1	O método *isdigit()* definido para os objetos da classe String, testa se uma string contém apenas dígitos. Dada uma string s = "534" e uma lista com apenas um elemento l = ["534"], ao escrevermos s.isdigit() recebemos como resposta o booleano True, indicando que a string s contém apenas dígitos. Por outro lado, ao escrevermos l.isdigit(), a resposta obtida será: a) Um erro, pois o objeto list não tem um método "isdigit" definido em sua classe. b) True, pois os elementos de l contêm apenas dígitos c) False, pois uma lista não pode ser convertida para um dígito. d) False, pois a lista é composta de uma string.
Nível 2	O que acontece com as duas linhas de código abaixo? a = 3 a.append(2)

	a) um erro, pois a variável a não é uma lista, portanto não possui uma implementação de *append()*. b) a variável a conterá o valor 5. c) a variável a conterá a lista [3, 2]. d) a variável a conterá a lista [2, 3].

Nível 3	Qual será a saída do programa abaixo?
	`class A:`
	`  i = 10`
	`  def fuuA(self):`
	`  print(self.i)`
	
	`class B:`
	`  i = 20`
	`  def fuuB(self):`
	`  print(self.i)`
	
	`a = A()`
	`b = B()`
	`i = 30`
	`a.fuuB()`
	
	a) Um erro pois o objeto *a* não tem um método *fuuB()* definido em sua classe.
	b) 10
	c) 20
	d) 30

PH.2

Código:	PH.2
Nome:	Uso de um método que retorna uma nova instância do mesmo objeto como se esse método alterasse a própria instância
Nível 1	O método *lower()* definido para strings, retorna uma string com o mesmo conteúdo da string para a qual esse método foi chamado, tal que todos os caracteres maiúsculos sejam minúsculos. Sendo assim pode-se afirmar que esse método: a) não altera string para a qual foi chamado a menos que essa seja sobrescrita pelo retorno da função. b) altera a string para a qual foi chamado, fazendo com que todos os caracteres

	se tornem minúsculos. c) tem efeito somente dentro da função print(). d) altera qualquer string recebido como argumento.
Nível 2	O que acontece ao se executar as linha de código abaixo? a = "bola carro" a.replace("bola", "casa") a) a string *a* conterá “bola carro” b) a string *a* conterá “casa carro” c) a string *a* conterá ”carro casa” d) a string *a* conterá “bola carro casa”

Nível 3	Qual será a saída do programa abaixo?
	
	class frase:
	conteudo = ""
	def __init__(self, f):
	self.conteudo = f
	def sem_primeira_palavra(self):
	a = self.conteudo
	nova_frase = frase(' '.join(a[1:]))
	return nova_frase
	minha_frase = frase(["cachorro", "bola", "gato"])
	minha_frase.sem_primeira_palavra()
	print(minha_frase.conteudo)
	
	a) ['cachorro', 'bola', 'gato']
	b) ['bola', 'gato']
	c) ['cachorro', 'bola']
	d) ['cachorro', 'gato']

PH.3

Código:	PH.3
Nome:	Função chamada após uma partícula *def.*
Nível 1	Qual das seguintes alternativas apresenta uma sintaxe correta para **chamar** (usar) uma função *fuu*? a) fuu() b) def fuu()

	c) fuu(): d) def fuu():
Nível 2	Considere a função a seguir: def prod(a, b): c = a * b return c Qual das alternativas abaixo calcula corretamente o produto de 10 e 3? a) prod(10, 3) b) def prod(10, 3) c) def prod(10, 3): d) prod(10, 3):
Nível 3	Através do programa abaixo, um aluno de programação tentou imprimir na tela o número 11, mas o programa abaixo não é capaz de fazê-lo pois possui um erro. Em qual das linhas desse programa está este erro? 1. def soma3(a): 2. b = a+3 3. return b 4. 5. d = 5 6. def soma3(d) 7. d = soma3(d) 8. print(d) a) 6 b) 7 c) 1 d) 2

PH.4

Código:	PH.4
Nome:	Atributo invocado sem ser previamente importado e sem, especificação de classe.
Nível 1	Para utilizarmos o método *sqrt()* da biblioteca *math* devemos: a) importar a biblioteca e utilizá-la como um atributo da classe importada: *math.sqrt()*. b) chamar a função pelo mesmo nome durante o programa: *sqrt()*

	c) importar a biblioteca e chamar a função pelo mesmo nome durante o programa: *sqrt()* d) apenas utilizá-la como um atributo da classe importada: *math.sqrt()*.
Nível 2	Um estudante de programação gostaria de calcular a raiz quadrada de um um número e para isso escreveu o seguinte código: num = 54756 raiz = "?" print(raiz) Para que esse código funcione devemos: a) igualar a raiz a math.sqrt(num) e importar a biblioteca math. b) igualar a raiz a sqrt(num). c) igualar a raiz a sqrt(num) e importar a biblioteca math. d) igualar a raiz a math.sqrt(num).
Nível 3	Qual a saída do programa abaixo? a = 60 b = 40 c = a+b d = sqrt(a+b) print("d = ", d, "=", "raiz de", a, "+", b) a) um erro pois a biblioteca math não foi importada e o atributo math não foi chamado pela classe math. b) d = 10.0 = raiz de 60 + 40. c) d = 10.0 = raiz de 100. d) um erro pois a biblioteca math não foi importada.

PH.5

Código:	PH.5
Nome:	Atribuição de valor a um método.
Nível 1	O método append() definindo para a classe de listas introduz um novo elemento ao de fim de uma lista pré-existente, ou uma lista vazia. Para utilizar esse método para qualquer elemento, devemos: a) passar o elemento como parâmetro para o método append aplicado à lista. b) atribuir o método append, aplicado à lista, a esse elemento. c) atribuir o método append a esse elemento. d) atribuir a lista a esse elemento.

Nível 2	Deseja-se que o último elemento das listas abaixo seja a soma dos outros dois. Observe o seguinte código: a = [1, 2] b = [3, 4] c = [5, 6] d = [7, 8] a.append(a[0]+a[1]) b.append = b[0] + b[1] c.append() = c[0] + c[1] append() = d[0] + d[1] Esse objetivo somente será alcançado para a lista: a) a b) b c) c d) d

Nível 3	A classe abaixo tem um nome como atributo, e um método que verifica se uma letra pertence a esse nome:
	
	class Palavra:
	def __init__ (self, nome):
	self.nome = nome
	def tem_a_letra(self, letra):
	if len (letra) != 1:
	(letra, "não é uma letra")
	elif letra in self.nome:
	print (self.nome, "tem a letra", letra)
	else:
	print(self.nome, "não tem a letra", letra)
	
	Ao se executar as linhas abaixo, teremos como saída:
	
	palavra = Palavra("cachorro")
	palavra.tem_a_letra = "d"
	
	a) A saída será vazia
	b) cachorro não tem a letra d
	c) cachorro tem a letra d
	d) d não é uma letra

PH.6

Código:	PH.6
Nome:	Ausência da partícula *self* para fazer referência ao atributo de uma instância.
Nível 1	Ao criarmos uma classe em Python, muitas vezes precisamos criar métodos que, por sua vez, utilizam dos atributos da própria classe. Para fazer referência a esses atributos, devemos utilizar a partícula: a) self b) nenhuma partícula é necessária c) get d) int

Nível 2	Dada uma classe com apenas um número como atributo, deseja-se criar um método que o imprima na saída padrão. Observe:
	`class Numero:`
	`  num = 3`
	`  def imprime(self):`
	`  print("COMPLETE AQUI")`
	
	Para isso, a string "COMPLETE AQUI" deve ser substituída por:
	a) self.num
	b) num
	c) Numero
	d) a string deve ser apenas deletada

Nível 3	Em um jogo implementado em Python, foi criada uma Classe para um personagem, como demonstrado no programa abaixo:
	
	`1. class Mago:`
	`2.  pontos_de_vida = 100`
	`3.  ataque = "RAIO ZAP"`
	`4.  pontos_de_ataque = 40`
	`5.  def atacar(self, inimigo):`
	`6.  inimigo.pontos_de_vida -= COMPLETAR`
	`7.  print(COMPLETAR)`
	
	`alberto = Mago()`
	`astolfo = Mago()`
	`astolfo.atacar(alberto)`
	
	Nesse Jogo, quando o personagem Astolfo ataca Alberto, os pontos de vida de Alberto são reduzidos pelos pontos de ataque de Astolfo. Além disso, a frase de ataque de alberto é imprimida na saída padrão. Qual alternativa completa

	corretamente os códigos faltantes acima? a) self.pontos_de_ataque na linha 6 e self.ataque na linha 7 b) pontos_de_ataque na linha 6 e ataque na linha 7 c) self.pontos_de_ataque na linha 6 e ataque na linha 7 d) pontos_de_ataque na linha 6 e self.ataque na linha 7

PH.7

Código:	PH.7
Nome:	Tentativa de acessar atributos de uma classe através de índices.
Nível 1	Para acessar os atributos de uma classe devemos: a) chama-lo pelo nome através do objeto. b) chamá-lo pelo índice i entre colchetes ”[i]” em ordem de atribuição. c) chamá-lo pelo índice i entre colchetes ”[i]” em ordem alfabética. d) chamá-lo pelo índice i entre chaves “{i}” em ordem de atribuição.
Nível 2	Dada a seguinte classe: class minhaClasse: nome = "Assis" profissao = "escritor" pessoa = minhaClasse() para imprimirmos na saída padrão o nome de *pessoa* seguido se sua profissão devemos usar: a) print(objeto.nome, objeto.profissao) b) print(objeto[0], objeto[1]) c) print(objeto[1], objeto[2]) d) print(objeto{0}, objeto{1})

Nível 3	Observe o programa abaixo que trabalha com uma lista de compras:
	
	class Compras:
	produtos = []
	precos = []
	
	def __init__(self, compra, valor):
	self.produtos.append(compra)
	self.precos.append(valor)

	def add_produto(self, compra, valor):
	  self.produtos.append(compra)
	  self.precos.append(valor)
	
	l = Compras("miojo", 3.00)
	l.add_produto("sabão", 5.00)
	l.add_produto("farinha", 2.50)
	l.add_produto("refrigerante", 5.00)
	
	Para acessar o preço da farinha, deve-se escrever:
	a) l.precos[2]
	b) l[1][2]
	c) l[2][2]
	d) l{2}[2]