Ficha da Unidade

Mestrado Integrado em Engenharia Informática e Computação
Fundamentos da Programação
Ocorrência: 2019/2020
—

Institutional page

Informação geral

Unidade: Fundamentos da Programação
Código: EIC0005
Sigla: FPRO
Curso: MIEIC
Anos Curriculares: 1º
Página oficial: Sigarra
Créditos ECTS: 6
Horas de Contacto: 70
Horas Totais: 162
Ocorrência: 2019/2020
Semestre: 1S
Teóricas: 2x1,5h: João Correia Lopes (JCL)
Teórico-Práticas: 8x2h: João Correia Lopes (JCL), Rui Camacho (RCS), Ricardo Cruz (RPMC), Fernando Cassola Marques (FJCM)
Attendance: 2x1h: António Cadilha, Telmo Baptista

Língua de trabalho

Português.

Objetivos

1. INTRODUÇÃO

A fluência no processo de desenvolvimento de software é um pré-requisito básico para o trabalho dos engenheiros de informática. Para usar computadores na resolução de problemas de maneira eficaz, os estudantes devem ser competentes na leitura e escrita de programas usando linguagens de programação de Alto Nível.

2. OBJETIVOS ESPECÍFICOS

O objetivo global desta Unidade é dar ao estudante a capacidade de criar algoritmos e de usar uma linguagem de programação para implementar, testar e depurar algoritmos para resolver problemas

simples.

O estudante será capaz de entender e usar os conceitos fundamentais de programação e a abordagem funcional da programação, especificamente a programação livre de efeitos (*effect-free*), onde as chamadas de função não têm efeitos colaterais e as variáveis são imutáveis, e de contrastar esta abordagem com a abordagem imperativa.

3. DISTRIBUIÇÃO PERCENTUAL

Componente científica: 40%
Componente tecnológica: 60%

Resultados de aprendizagem e competências

No final da unidade, espera-se que o estudante consiga resolver problemas através do desenvolvimento de programas de complexidade média, usando as abordagens ou paradigmas de programação imperativa e funcional.

Mais especificamente, o estudante deverá ser capaz de:

1. Projetar, implementar, testar e depurar um programa que use as construções fundamentais de programação, cálculos básicos, estruturas condicionais e iterativas padrão, entrada/saída simples, persistência (ficheiros) e exceções.
2. Entender a abstração de dados e usar tipos de dados simples ou compostos.
3. Entender a abstração procedimental e usar a definição de funções, passagem de parâmetros, recursão.
4. Implementar algoritmos básicos que evitem atribuir a um estado mutável ou considerar igualdade de referência.
5. Escrever funções úteis que aceitem e devolvam outras funções (funções de ordem mais elevada).
6. Entender variáveis e escopo léxico num programa (incluindo *closures*).
7. Definir e usar iteradores e outras operações em coleções, incluindo operações que tomam funções como argumentos (*map*, *reduce/fold* e *filter*) e compreensões em listas.
8. Usar as ferramentas de programação que ajudam a escrever, testar e documentar programas em computador de acordo com as melhores práticas de programação.

Modo de trabalho

Presencial.

Pré-requisitos (conhecimentos prévios) e co-requisitos (conhecimentos simultâneos)

Não são necessários conhecimentos prévios.

Programa

- Introdução ao Pensamento Computacional, algoritmos e programação com Python.
- Conceitos fundamentais de programação: tipos de dados simples; variáveis, expressões e declarações; fluxo de programa, condicionais, iteração; funções, passagem de parâmetros, recursão; persistência.
- Tipos de dados compostos: strings, tuplos, conjuntos, listas, dicionários.
- Programação livre de efeitos usando chamadas a função sem efeitos colaterais e variáveis imutáveis.
- Funções como cidadãos de primeira classe e *closures*.
- Funções de ordem superior: *map*, *reduce/fold* e *filter*.
- Geradores e compreensões em listas.
- Estratégias de resolução de problemas.
- Ferramentas de programação, teste e depuração.

Bibliografia Obrigatória

- Peter Wentworth, Jeffrey Elkner, Allen B. Downey, and Chris Meyers, *How to Think Like a Computer Scientist — Learning with Python 3*, Release 3rd Edition, 2019 [PDF] [HTML] [Biblioteca]

Bibliografia Complementar

- Steven F. Lott, Building Skills in Python - A Programmer's Introduction to Python, FreeTechBooks, 2010 HTML
- Allen Downey, Think Python — How to Think Like a Computer Scientist, 2nd Edition, Version 2.2.23, Green Tea Press, 2015 HTML PDF code
- David Mertz, Functional Programming in Python, O'Reilly Media, 2015 PDF
- Ernesto Costa, Programação em Python – Fundamentos e Resolução de Problemas. FCA – Editora de Informática, 2015. ISBN: 978-972-722-816-4 [Biblioteca]

Métodos de ensino e atividades de aprendizagem

O envolvimento contínuo do estudante com a unidade é promovido, através do estudo e discussão dos tópicos, distribuídos previamente em Notebooks Jupyter, tanto em aulas teóricas e teórico-práticas, como através de trabalhos de programação dentro e fora das aulas.

O estudante é encorajado a encontrar as melhores ideias para resolver problemas específicos, executá-las e implementar as soluções de programação, de forma elegante, legível e eficiente (em tempo e em espaço) usando a linguagem de programação Python.

São usadas ferramentas de correção automática de código para aumentar a rapidez do feedback dado aos estudantes.

As aulas teóricas (T) são usadas para apresentar e discutir os tópicos do programa, usando um computador ligado a um projetor multimédia.

As aulas teórico-práticas (TP) são usadas para ajudar os estudantes a entender os tópicos do programa e a resolver as tarefas de programação semanais.

As tarefas de programação em aula teórica e fora das aulas são dadas, semanalmente, para melhorar o desenvolvimento regular e eficaz dos processos de aprendizagem individual autónomo e são avaliadas em tarefas do Moodle.

Tarefas em sala de aula são usadas para avaliação somativa no final de cada aula teórica através de questionários no Moodle.

Os estudantes são incentivados a usar uma App Web (Play) contendo exercícios de programação selecionados por tema: exercícios (fáceis) para realizar antes da aula teórica, exercícios (dificuldade média) para realizar antes das tarefas semanais (RE) e exercícios (difíceis) para realizar antes das provas práticas em computador (PE).

No trabalho em sala de aula, os estudantes usam o mesmo ambiente de trabalho (IDE Spyder, Pylint, App Web (Test) e de submissão de exercícios de programação) que é usado posteriormente nas avaliações individuais (PE).

Sempre que for considerado necessário na sala de aula, especialmente durante o primeiro mês de aulas da unidade, os alunos são incentivados a passar pela "Clínica", assegurada por monitores, para obter ajuda.

Software

- Anaconda Distribution 2019.03 for Linux (https://www.anaconda.com)
- Spyder3 (incluido em Anaconda)
- Python 3.7 (incluido em Anaconda)

Palavras-chave

Ciências Físicas > Ciência de computadores > Programação

Tipo de avaliação

Avaliação distribuída sem exame final.

Avaliação e componentes de Ocupação

Descrição	Tipo	Tempo (horas)	Data de Conclusão
Participação presencial (estimativa)	Aulas	70	
PE01	Teste/Exame	0	16/10/2019
PE02	Teste/Exame	0	06/11/2019
PE03	Teste/Exame	0	27/11/2019
PE04	Teste/Exame	0	18/12/2019

Descrição	Tipo	Tempo (horas)	Data de Conclusão
PE05	Teste/Exame	0	08/01/2020
TE01	Teste/Exame	0	15/01/2020
Estudo	Estudo	92	
	Total:	162	

Obtenção de frequência

Os estudantes são admitidos ao teste teórico (TE), se não excederem o limite de faltas (25% do número total de aulas teórico-práticas estimadas) e se obtiverem um mínimo de 40% pelo menos numa das duas últimas avaliações práticas em computador (MAX(PE04, PE05) >= 40%).

Fórmula de cálculo da classificação final

A avaliação será baseada nas seguintes componentes:

LE = Perguntas de resposta múltipla, sobre conceitos de programação, realizadas individualmente no Moodle em sala de aula teórica (são selecionadas as melhores 20 classificações de um total de 26) *[5 perguntas, 5 minutos]*
RE = Exercícios de programação semanais para casa e nas aulas teórico-práticas (são selecionadas as melhores 10 classificações de 13) *[5 perguntas, NA]*
PE = Avaliação prática em computador a realizar, individualmente, no Moodle (são selecionadas as melhores 3 classificações de 5) *[5 perguntas, 105 minutos]*
TE = Avaliação teórica através de questionário de respostas múltiplas, a realizar individualmente no Moodle com consulta de um livro *[50 perguntas, 105 minutos]*

Classificação final = 10% LE + 10% RE + 50% PE + 30% TE

Observações:

1. É exigida classificação mínima de 40% na componente TE
2. É exigida a classificação mínima de 40% em pelo menos uma das 2 últimas provas (PE04 ou PE05)
3. Terá lugar uma segunda prova teórica (TE02) para estudantes admitidos e que faltem ao teste com justificação válida, ou de recuperação para os estudantes admitidos e que não obtiveram a nota mínima; no último caso, a nota nesta componente é limitada a 50%
4. Se a classificação final for maior que 17, o estudante pode ser submetido a uma avaliação oral e a nota final é a média das duas classificações
5. Os estudantes que atingirem pelo menos 90% nas tarefas de avaliação presencial e de programação semanal (LE + RE), a meio do semestre podem optar por manter a nota e fazer um pequeno projeto para atingir a nota máxima (100%) nessas duas componentes

Provas e trabalhos especiais

A avaliação desta unidade usa três tipos de provas::

1. Exercícios em aula teórica, ou fora de aula, relacionadas com os tópicos atuais, a serem

submetidos no Moodle e classificados automaticamente
2. Avaliação prática em computador de um conjunto de pequenos programas ou funções, utilizando o Moodle e classificados automaticamente, e com consulta de textos de referência
3. Avaliação teórica através de um questionário de escolha múltipla no Moodle, com consulta de um livro, para avaliar a aprendizagem dos conceitos fundamentais de programação e a capacidade dos estudantes usarem e discutirem os melhores algoritmos e estruturas de dados para problemas específicos

Melhoria de classificação

No ano letivo seguinte, o estudante pode solicitar melhoria de classificação que inclui a avaliação prática em computador PE04 ou PE05 e a avaliação teórica TE01 ou TE02, com consulta de apenas um livro, igual à realizada pelos estudantes inscritos.

Classificação final = 60% PE + 40% TE

Observações:

1. É exigida classificação mínima de 40% em cada um dos componentes PE e TE
2. Se a classificação final for maior que 17, o estudante pode ser submetido a uma avaliação oral e a nota final é a média das duas classificações

Avaliação especial (TE, DA, ...)

No caso de estudantes que não frequentem as aulas e dispensem a avaliação semanal (LE e RE), a classificação final será a média pesada da classificação das restantes componentes de avaliação: avaliação prática em computacional (PE) e avaliação teórica (TE), realizadas tal como os estudantes ordinários.

Classificação final = 60% PE + 40% TE

Observações:

1. É exigida classificação mínima de 40% em cada um dos componentes PE e TE
2. É exigida a classificação mínima de 40% em pelo menos uma das 2 últimas provas (PE4 ou PE5)
3. Terá lugar uma segunda prova teórica (TE02) para estudantes admitidos e que faltem à avaliação com justificação válida, ou de recuperação para os estudantes admitidos e que não obtiveram a nota mínima. No último caso, a nota nesta componente é limitada a 50%
4. Se a classificação final for maior que 17, o estudante pode ser submetido a uma avalaição oral e a nota final é a média das duas classificações

Observações

Nenhuma classificação de componente individual (LE[1)], RE, PE, TE[2)]) pode ser reutilizada noutra inscrição na unidade curricular.

— FPRO, 2019/2020

1)

LE = Ler o livro (e perceber) + Ver bons exemplos de programação

2)

REPETE = Repetir o trabalho com programas e exercícios dados ou encontrados noutros lugares