



Instituto Politécnico da Guarda
Escola Superior de Tecnologia e Gestão

Caderno de exercícios
Programação em Python

18 de Outubro de 2020

Curso: Engenharia Informática

Unidade Curricular: Algoritmos e Estruturas de Dados

Ano Letivo: 2020/2021

Docente: Paulo Jorge Costa Nunes

Coordenador da área disciplinar: José Carlos Coelho Martins da Fonseca

Conteúdo

I	Python e Algoritmos	9
1	Introdução	11
2	Instalação e configuração do Python	13
2.1	Funcionalidades	15
2.2	Características	15
2.3	Elementos de um programa Python	16
2.3.1	Nomes	16
2.3.2	Expressões e operadores	17
2.3.3	Funções de saída	17
2.3.4	Atribuição simples	18
2.3.5	Atribuição múltipla	19
2.3.6	Entrada e atribuição	19
2.3.7	Tipos de dados	19
2.4	Biblioteca de funções matemáticas	20
2.5	Conversão e arredondamento entre tipos de dados	21
2.6	Acumulador	23
2.7	Função range	24
2.8	Função list	25
2.9	Strings	25
2.9.1	Padrão travessia	26
2.10	Objectos	28
2.11	Estruturas de repetição	29
2.12	Módulos	29
2.13	A estrutura e sintaxe do Python	29
2.14	Definição de constantes	29
2.15	Datas e horas	30
2.15.1	Número de dias até uma data	31
3	Operações com a Internet	33
3.1	Obter páginas web com <i>urlopen</i>	33
4	Cálculo de perímetros, áreas e volumes	35
4.1	Perímetro de um retângulo	35
4.1.1	Desenvolvimento do algoritmo	35
4.1.1.1	Modelo	35
4.1.1.2	Esboço	35
4.1.1.3	Algoritmo	36
4.1.2	Programa	37
4.1.2.1	Caso 1	37
4.2	Volume paralelepípedo	37

4.2.1	Desenvolvimento do algoritmo	38
4.2.1.1	Modelo	38
4.2.1.2	Esboço	38
4.2.1.3	Algoritmo	39
4.2.1.4	Programa	39
4.2.1.5	Caso 1	40
4.3	Proposta de exercícios	40
4.3.1	Perímetro de uma rotunda	40
4.3.2	Área de uma rotunda	40
4.3.3	Volume de um cilindro	40
5	Cálculo de médias e classificações qualitativas	41
5.1	Média de três números	41
5.2	Média de vários números	41
5.3	Classificação qualitativa	41
5.4	Qualidade do ar	41
6	Operações com data e horas	43
6.1	Contagem do número de dias de um dado dia da semana entre duas datas	43
6.2	Contagem do ocorrências entre duas datas com periodicidade	43
6.3	Número de dias até uma data	44
6.3.1	Geração de datas de eventos recorrentes	44
6.3.2	Eventos em tabela com nome do mês do dia da semana	44
6.3.3	Eventos em tabela na língua inglesa	45
6.3.3.1	Eventos em tabela numa dada língua	45
6.3.3.2	Eventos tabela HTML com interrupções	46
7	Operações com números	47
7.1	Decompor os dígitos de um número	47
8	Ciclos de repetição	49
8.1	Ciclo while	49
9	Estruturas de dados	51
9.1	7.3. struct ? Interpret strings as packed binary data	51
10	Dicionário em ficheiro	53
11	Tipos de dados	55
11.1	Calendários datas e horas	55
11.2	Calendar - General calendar-related functions	56

Lista de Figuras

2.1	Teste de instalação, janela shell interativa do Python.	13
2.2	Teste de instalação, janela shell interativa do Python.	14
2.3	Programa hello.py no editor Python.	14
2.4	Resultado do programa hello.py para o nome Ana.	15
2.5	Execução de um script Python em linha de comando MAC-OSX. Chamada e output.	16
2.6	Palavras reservados (31) do Python.	16
4.1	Modelo perímetro de um retângulo.	35
4.2	Caso 1: perímetro de um retângulo.	38
4.3	Modelo perímetro de um retângulo.	38
4.4	Caso 1: Programa Python volume de um paralelepípedo.	40
5.1	Classificação do Índice de Qualidade do Ar proposto para o ano 2013 ($\mu\text{g}/\text{m}^3$) .	42
6.1	Interface eventos entre duas datas com periodicidade	43
6.2	Eventos entre duas datas.	44
6.3	Eventos entre duas datas numa tabela HTML em português.	45
6.4	Eventos entre duas datas numa tabela HTML em inglês.	45

Listings

2.1	Programa hello.py	14
2.2	Script Python Hello.	16
2.3	Exemplos da instrução print.	18
2.4	Exemplos de atribuição de valores a variáveis.	19
2.5	Entrada e atribuição de valores a variáveis.	19
2.6	Exemplos de atribuição de valores a variáveis.	20
2.7	Python Shell. Determinação to tipo de diversas variáveis.	20
2.8	Programa quadratic.py	21
2.9	Python Shell. Exemplos de conversão entre tipos.	22
2.10	Python Shell. Exemplos de conversão entre tipos reais.	23
2.11	Cálculo do valor de 6!	23
2.12	forma geral de um algoritmo acumulador	24
2.13	Programa para calcular 6! utilizando um acumulador.	24
2.14	Exemplo da utilização da função range().	24
2.15	Programa para calcular o fatorial de um número utilizando for e range().	25
2.16	Factorial do número 100.	25
2.17	Exemplo da utilização da função list().	25
2.18	Exemplo de strings e operações.	26
2.19	Travessia com while.	27
2.20	Travessia com for.	27
2.21	Saída dos programas tavessia com while e for	28
2.22	Travessia com for.	28
2.23	Programa futval.py	29
2.24	Exemplos de definição de constantes.	29
2.25	Exemplos de definição de constantes.	29
2.26	Exemplos de definição de constantes em classes.	30
2.27	Exemplos de formatação de data e horas.	30
2.28	Número de dias até 4 de julho.	31
3.1	Codificação de parâmetros (<i>urlencode</i>)	34
4.1	Esboço do algoritmo perímetro de um retângulo.	36
4.2	Algoritmo perímetro de um retângulo sem validação de dados de entrada.	36
4.3	Algoritmo perímetro de um retângulo.	37
4.4	Programa Python perímetro de um retângulo	37
4.5	Esboço do algoritmo o volume de um paralelepípedoo.	39
4.6	Algoritmo Programa Python volume de um paralelepípedo	39
4.7	Programa Python volume de um paralelepípedo	40
6.1	Exemplos de formatação de data e horas.	43
6.2	44
6.3	Seleção de zona de tempo (Europe/Lisboa) e localidade (Portugal: pt_BR).	45
6.4	Seleção de zona de tempo (Europe/London) e localidade (en_GB).	45
8.1	Programa contador com ciclo while	49

8.2	Resultado do programa contador com ciclo while	49
-----	--	----

Parte I

Python e Algoritmos

Capítulo 1

Introdução

Com este caderno de exercícios pretende-se contribuir para o ensino da linguagem de programação Python na unidade curricular de Algoritmos e Estrutura de Dados do curso de Engenharia Informática.

Existem muitos recursos na Internet para desenvolvimento em Python, de seguida são apresentados alguns:

1. **Python** — Site oficial to Python <https://www.python.org>.
2. **python-course.eu** — <https://www.python-course.eu/index.php> — Os itens da linguagem Python são apresentados com muitas figuras. São também, apresentadas figuras com exemplos de aplicações reais relacionadas com os itens.¹
3. **EduMaven / Python Programming** (879 pages) — <https://edumaven.com/python-programming/>
4. **Python 3 Tutorial in PDF - TutorialsPoint** — http://www.tutorialspoint.com/python3/python3_tutorial.pdf.
5. **Python 3.5.2 documentation** — <https://docs.python.org/3.5/>.
6. **The Python Package Index (PyPI)** — O PyPI é um repositório de software para a linguagem de programação Python. Atualmente (2016-09-12) com 88441 módulos desenvolvidos por terceiros <https://pypi.python.org/pypi>.
7. **Python** — <https://www.python.org>.
8. **Python** — <https://www.tutorialspoint.com/python/>.
9. **Book** — <http://www.openbookproject.net/thinkcs/python/english2e/>.
10. **How to Think Like a Computer Scientist: Learning with Python 2nd Edition documentation** — <http://www.openbookproject.net/thinkcs/python/english2e/#>.
11. **Book** — Python Programming: An Introduction to Computer Science by John M. Zelle, Ph.D <http://mcsp.wartburg.edu/zelle/python/>.

- Programas Simples — Diretórios com os programas <http://mcsp.wartburg.edu/zelle/python/ppics3/code/>.

¹Novo em 2018/2019

- Código fonte dos programas para download — <http://mcsp.wartburg.edu/zelle/python/ppics3/code.zip>.
 - **Slides** — Slides Powerpoint para as aulas <http://mcsp.wartburg.edu/zelle/python/ppics3/slides/>.
12. **Book** - Practical Programming (2nd edition) An Introduction to Computer Science Using Python 3 by Paul Gries, Jennifer Campbell, Jason Montojo — <https://pragprog.com/book/gwpy2/practical-programming>
- Programas Simples — Diretórios com os programas <http://mcsp.wartburg.edu/zelle/python/ppics3/code/>.
 - Código fonte dos programas para download — <http://mcsp.wartburg.edu/zelle/python/ppics3/code.zip>.
 - **Slides** — Slides Powerpoint para as aulas <http://mcsp.wartburg.edu/zelle/python/ppics3/slides/>.
13. **Book** — Invent Your Own Computer Games with Python, 2nd Edition 2nd Edition by Al Sweigart (Author) <http://inventwithpython.com/downloads/>.

Capítulo 2

Instalação e configuração do Python

No endereço <http://www.python.org/download/> estão disponíveis diversas versões do Python para diversos sistemas operativos/servidores web. Assim como as respectivas instruções de instalação e configuração do Python.

Os passos para instalar o Python no *Windows* são:

1. Visitar o endereço <http://www.python.org/download/> e descarregar (fazer o download) da versão mais atual do Python 3.
2. Double-click no ficheiro *.msi* descarregado e seguir as instruções. Todas as opções por omissão são adequadas para aprendizagem da linguagem por novos programadores.
3. Testar a instalação — Iniciar o *Python Shell* através do menu *Start* do Windows. Deverá aparecer uma janela semelhante à apresentada na figura 2.1 que tem o nome de *Python Shell*. O pronto (*prompt*) `> > >`, aguarda a escrita, a partir teclado, de operações do utilizador. As linhas `> > > 2 + 4`, `6` e `> > >` representam o resultado da operação de somar os números 2 e 4.

A figura 2.2 mostra a janela do editor IDLE (*Integrated DeveLopment Environment*) do Python semelhante à janela *Shell*. Esta janela apresenta uma barra de menus com diversas funcionalidades para editar, executar e fazer *debug* dos programas.

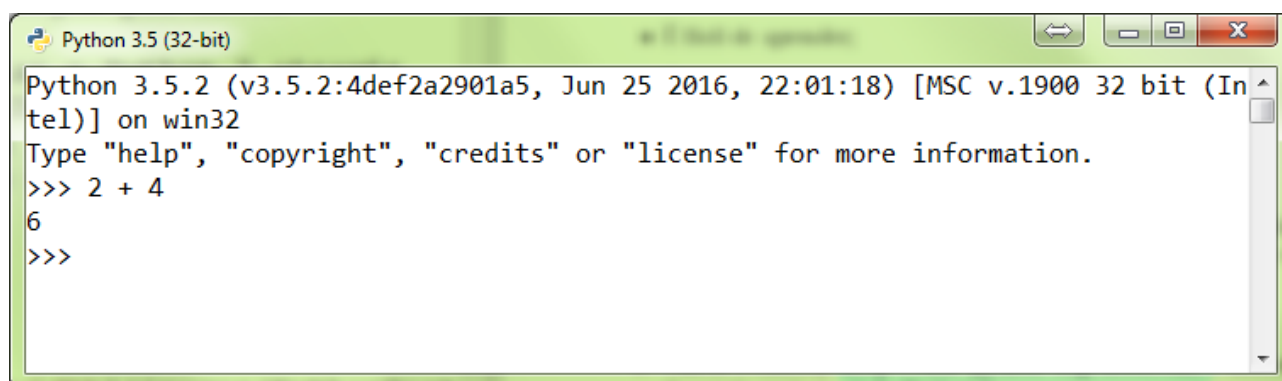


Figura 2.1: Teste de instalação, janela shell interativa do Python.

A listagem 2.1 mostra o código fonte de um programa escrito em Python. A linha 1 iniciada pelo símbolo `#` é um comentário em Python. As linhas de código 2 e 3 escrevem no ecrã e em linhas diferentes o texto contido entre os símbolos `'` (delimitadores de strings). A linha 4 lê a sequência de caracteres (nome) digitados pelo utilizador e armazena-os na variável `myName`. A

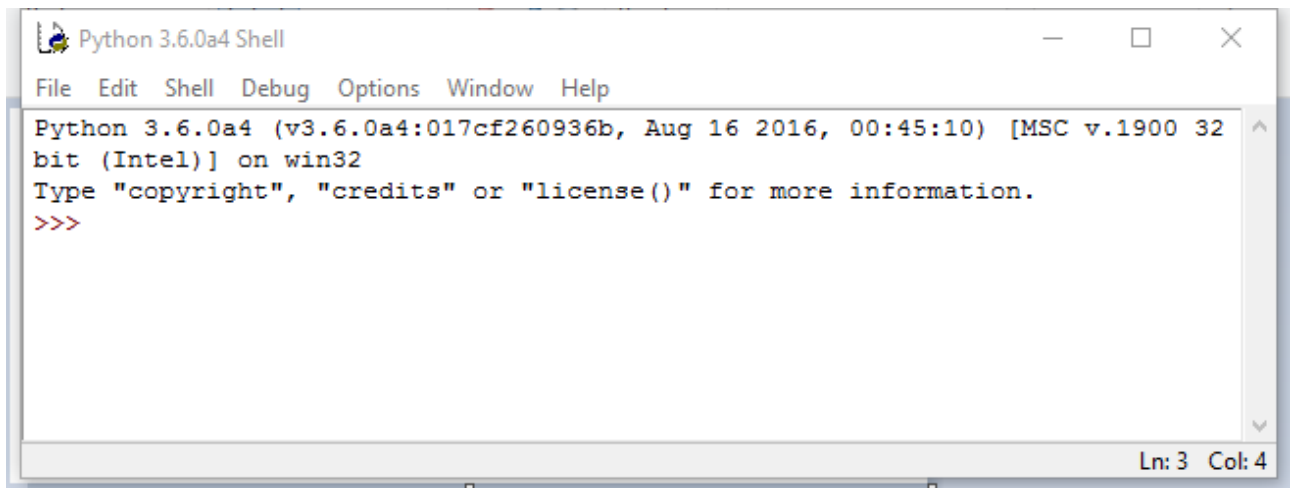
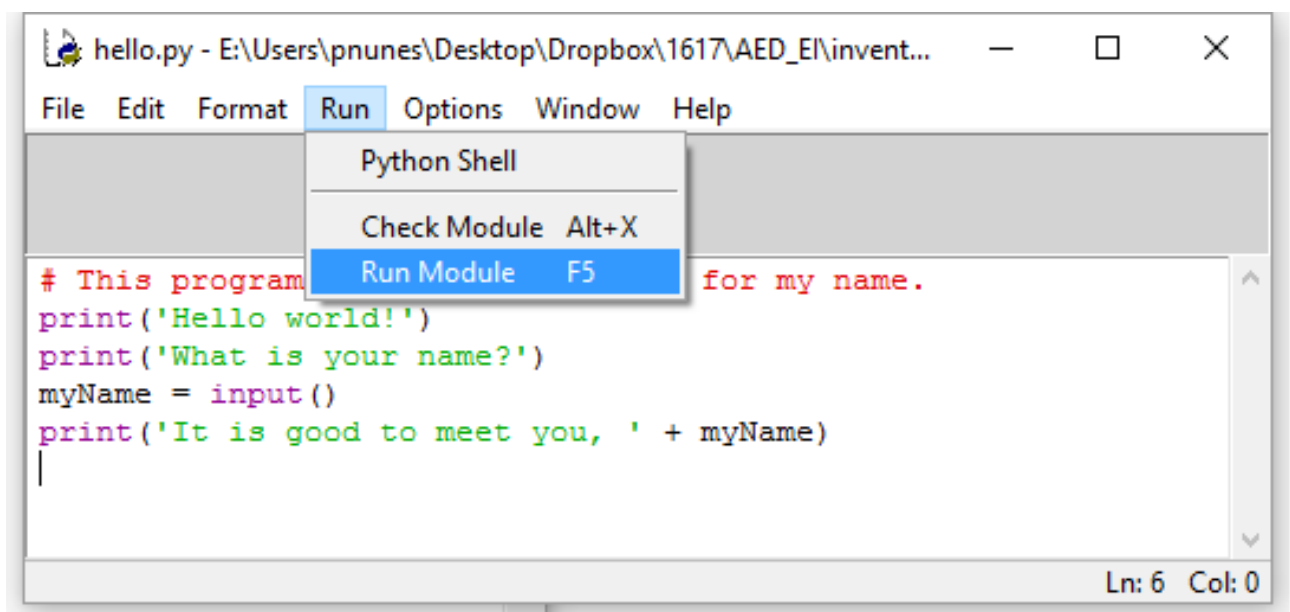


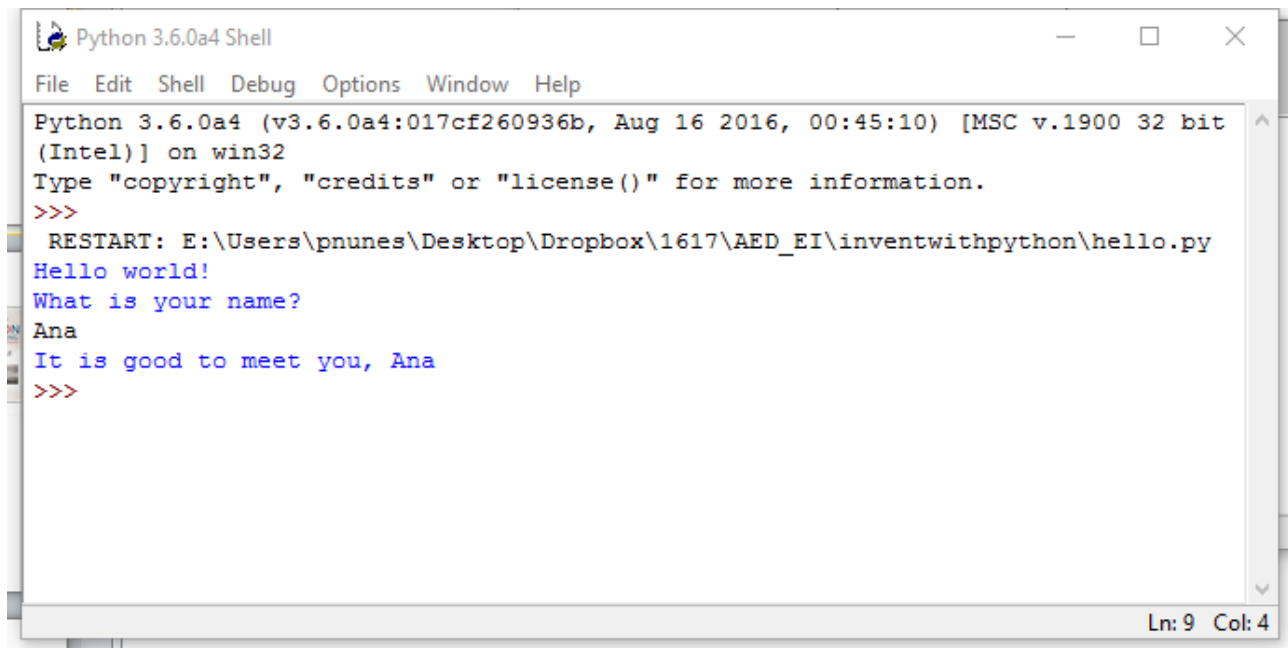
Figura 2.2: Teste de instalação, janela shell interativa do Python.

linha final escreve numa nova linha do ecrã o texto 'It is good to meet you, ', concatenado com o valor da referida variável (Ou seja o nome do utilizador).

As figuras 2.3 e 2.4 ilustram o programa `hello.py` no editor do Python e o output da sua execução com o Ana, respetivamente. A execução de programas pode ser efetuada através da opção *Run Module* do menu *Run*, como ilustra a 2.3.

```
1 # This program says hello and asks for my name.  
2 print('Hello world!')  
3 print('What is your name?')  
4 myName = input()  
5 print('It is good to meet you, ' + myName)
```

Listing 2.1: Programa `hello.py`Figura 2.3: Programa `hello.py` no editor Python.



```
Python 3.6.0a4 Shell
File Edit Shell Debug Options Window Help
Python 3.6.0a4 (v3.6.0a4:017cf260936b, Aug 16 2016, 00:45:10) [MSC v.1900 32 bit
(Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>>
RESTART: E:\Users\pnunes\Desktop\Dropbox\1617\AED_EI\inventwithpython\hello.py
Hello world!
What is your name?
Ana
It is good to meet you, Ana
>>>
```

Figura 2.4: Resultado do programa hello.py para o nome Ana.

2.1 Funcionalidades

O Python é uma linguagem extremamente modular ideal para ser instalado em servidores web na maioria dos sistemas operativos. Tem um conjunto enorme de funcionalidades, como por exemplo:

1. Desenvolvimento para a Web e Internet — Django, Pyramid, django CMS, HTML, XML, E-mail, FTP, IMAP, socket interface, HTTP client, etc.
2. Computação científica e numérica — SciPy, Pandas, IPython.
3. Acesso a bases de dados
4. Desenvolvimento de GUIs (Graphical User Interfaces) para Desktop — wxWidgets, Kivy, Qt via pyqt or pyside.
5. Educação — Python é uma linguagem excelente para ensinar programação, tanto a nível introdutório e em cursos mais avançados.
6. Programação com redes
7. Software & Game Development

2.2 Características

Algumas das principais características da linguagem de programação Python são listadas de seguida:

- O Python é potente ... e rápida;
- Pode ser executadas em qualquer sistema (computador/sistema operativo).
- É fácil de aprender;
- É open source.

2.3 Elementos de um programa Python

O Python pode ser executado na linha de comando de qualquer computador que tenha o Python instalado. Na listagem 2.2 é apresentado um script Python que escreve a mensagem "Hello from Python" utilizando o construtor de linguagem **echo**.

A figura 2.5 mostra um terminal com a execução e output do script **firstPythonScript.Python** em linha de comando através do comando bash **Python firstPythonScript.Python**

Listing 2.2: Script Python Hello.

```
print ("Hello_from_Python.")
```

Figura 2.5: Execução de um script Python em linha de comando MAC-OSX. Chamada e output.

2.3.1 Nomes

Os nomes são dados a variáveis e módulos (secção 2.12). As linguagens de programação permitem manipular (atribuir e ler) variáveis que é uma das mais poderosas funcionalidades das linguagens de programação.

Os nomes das variáveis e módulos são chamados de identificadores. Cada identificador deve começar com uma letra ou um *underscore* (`_`), seguido de qualquer sequência de letras, dígitos ou *underscores*. O seu comprimento é arbitrário e não podem ser usados os nomes apresentados na 2.6 porque são palavras reservadas (ou *keywords*) do Python. Isso significa que eles não estão disponíveis para ser usados como um nome para uma variável/módulo no programa.

Os identificadores são *case sensitive*. Ou seja, o identificador **preco** é diferente do identificador **Preco**.

and	as	assert	break	class	continue
def	del	elif	else	except	exec
finally	for	from	global	if	import
in	is	lambda	not	or	pass
print	raise	return	try	while	with
yield					

Figura 2.6: Palavras reservadas (31) do Python.

Exemplos de identificadores válidos e diferentes:

- mensagem
- n
- pi
- numero_filhos

- X
- Celsius
- celsius
- cElsius

Exemplos de identificadores inválidos:

- and — Palavra reservada.
- 2peso — Não pode começar por um número.

2.3.2 Expressões e operadores

Uma expressão pode ser:

1. Um fragmento de código que produz ou calcula novos valores de dados.
2. Identificadores simples também podem ser expressões.
3. Literais — usados para representar um valor específico, por exemplo 3.9, 1, 1.0.
4. Dados textuais (strings) — Exemplo: "Olá".

As expressões simples podem ser combinadas utilizando operadores aritméticos (tabela 2.1), lógicos (tabela 2.2) e de manipulação de bits (tabela 2.3).

Tabela 2.1: Operadores aritméticos

Operador	Nome	Exemplo
+	adição	$2 + 3 = 5$
-	subtração	$2 - 3 = -1$
*	multiplicação	$2 * 3 = 6$
//	divisão inteira (divisão inteira trunca o resultado)	$5 / 2 = 2$
/	divisão real	$5 // 2 = 2.5$
%	módulo (resto)	$5 \% 4 = 1$
**	exponenciação	$2.0 \hat{=} 3.0 = 8$
/	raiz quadrada	$ / 25.0 = 5$
/	raiz cúbica	$ / 27.0 = 3$
!	fatorial	$5 ! = 120$
!!	fatorial (operador de prefixo)	$!! 5 = 120$

2.3.3 Funções de saída

As funções de saída (output statements) permitem apresentar os resultados de programa. A função de saída `print()` permite imprimir qualquer número de expressões na mesma linha e tem a seguinte sintaxe:

```
print(<expr>, <expr>, ?, <expr>)
```

Instruções de saída `print(...)` sucessivas imprimem em linhas separadas. A instrução `print()` sem argumentos imprime uma linha em branco. A listagem 2.3 mostra a instrução `print()` com diferentes argumentos e tipos de dados.

Tabela 2.2: Operadores lógicos

Operador	Nome
and	E lógico
or	OU lógico
not	Negação
<	Menos de
>	Maior que
<=	Menor ou igual a
>=	Maior ou igual a
==	Igual a
!=	Não igual a

Tabela 2.3: Operadores para manipulação de bits

Operador	Nome	Exemplo
&	AND bit a bit	91 & 15 = 11
	OR bit a bit	32 3 = 35
^	XOR bit a bit	5 ^ 3 = 6
	NOT bit a bit	1 = -2
«	deslocamento à esquerda bit a bit	1 « 4 = 16
»	deslocamento à direita bit a bit	8 » 2 = 2

Listing 2.3: Exemplos da instrução print.

```
print(3+4)
print(3, 4, 3+4)
print()
print(3, 4, end=" "),
print(3 + 4)
print("The answer is ", 3+4)
```

Tabela 2.4: Saídas dos exemplos da listagem 2.3.

Instrução	Saída
>>> print(3+4)	7
>>> print(3, 4, 3+4)	3 4 7
>>> print()	
>>> print(3 + 4)	7
>>> print("The answer is", 3+4)	The answer is 7

2.3.4 Atribuição simples

Para atribuir valores a variáveis é utilizado o símbolo igual (=). A sintaxe é a seguinte:

<Variável> = <expr>

Onde **Variável** é um identificador e **expr** é uma expressão. Esta é avaliada para produzir um valor que é então associado com a variável.

Listing 2.4: Exemplos de atribuição de valores a variáveis.

```
x = 3.9 * x * (1-x)
fahrenheit = 9/5 * celsius + 32
x = 5
peso = 67.5
nome = "Ana"
curso = 'EI'
```

2.3.5 Atribuição múltipla

Em Python podem ser avaliadas várias expressões e atribuídos a vários variáveis ao mesmo tempo. A sintaxe é a seguinte:

```
<Var>, <var>, ... = <expr>, <expr>, ...
```

O exemplo: `x, y = x, y`, ao contrário de outras linguagens de programação (C, C++, Java, PHP, JavaScript) permite trocar o valor de duas variáveis sem a necessidade de usar uma variável auxiliar.

2.3.6 Entrada e atribuição

A função de entrada `input()` permite fazer a entrada de dados do utilizador (teclado). A sintaxe da função para leitura de número é a seguinte:

```
<Variável> = eval (input (<prompt>))
```

A função `eval()` avalia a cadeia de caracteres digitados pelo utilizador e transforma-o num valor (número em Python). A expressão `prompt` é uma cadeia de caracteres que representa uma mensagem a apresentar ao utilizador.

Para entrada de strings a sintaxe é a seguinte:

```
<Variável> = input (<prompt>)
```

Na listagem 2.5 são apresentados vários exemplo de entrada e atribuição de valores a variáveis de diversos tipos (números inteiros, número e reais cadeia de texto).

Listing 2.5: Entrada e atribuição de valores a variáveis.

```
nome = input("Nome?")
idade = eval(input("Idade?"))
peso = eval(input("Peso (Kg)?"))
```

2.3.7 Tipos de dados

No Python não é necessário declarar as variáveis. O tipo da variável depende do valor que armazena. O tipo da variável pode mudar várias vezes durante a execução do programa. Os tipos de variáveis básicos em Python são número inteiros, números reais e strings. A listagem 2.9 apresenta vários exemplos.

Listing 2.6: Exemplos de atribuição de valores a variáveis.

```

vencimento_base = 3437.34
pi = 3.14159
raiz_de2 = 1.4142124209
e = 2.71828
polegada = 2.54
grau_celcius = 32.0
velocidade_luz = 299792458
velocidade_som = 343.4
nome = "Carlos"
marca = "BMW"
marca = 1

```

O tipo de dados de uma variável determina quais os valores que ele pode ter e quais as operações que podem ser executadas nela. Em Python existe uma função especial para que permite obter o tipo de qualquer variável.

Listing 2.7: Python Shell. Determinação to tipo de diversas variáveis.

```

>>> type(3)
<class 'int'>
>>> type(3.1)
<class 'float'>
>>> nome = "Ana"
>>> type(nome)
<class 'str'>
>>>

```

2.4 Biblioteca de funções matemáticas

Uma biblioteca é um módulo (secção 2.12) com algumas definições e/ou funções de utilização geral. Ou seja, comum a vários programas. Na tabela 2.5 são mostradas algumas funções matemáticas da biblioteca *math* do Python.

Para usar as funções definidas numa biblioteca, é necessário incluí-la no programa utilizando a palavra reservada *import* do Python com a seguinte sintaxe:

```
import <nome_biblioteca>
```

Para importar a biblioteca de matemática *math* tem de ser incluída a seguinte linha de código no início do programa:

```
import math
```

A listagem 2.8 ilustra a utilização da função matemática raiz quadrada, *sqrt()*, para calcular as soluções da equação do 2º grau. Note-se que a referida função, na linha 14, está precedida do nome da biblioteca *math*.

Tabela 2.5: Funções matemáticas no Python.

Função/definições	Descrição	Exemplo/valor
pi	Uma aproximação do pi	3.141592653589793
e	Uma aproximação do e	2.718281828459045
abs()	valor absoluto	abs(-5.0) = 5
sqrt(x)	A raiz quadrada de x	
sin(x)	O seno de x	
cos(x)	O co-seno de x	
tan(x)	A tangente de x	
asin(x)	O inverso do seno x	
acos(x)	O inverso do cosseno x	
atan(x)	O inverso da tangente x	
log(x)	The natural (base e) logarithm of x	
log10(x)	The common (base 10) logarithm of x	
exp(x)	The exponential of x	
ceil(x)	The smallest integer not less than x	
floor(x)	The largest integer not greater than x	

```

1 # quadratic.py
2 #   A program that computes the real roots of a quadratic equation.
3 #   Illustrates use of the math library.
4 #   Note: This program crashes if the equation has no real roots.
5
6 import math # Makes the math library available.
7
8 def main():
9     print("This program finds the real solutions to a quadratic")
10    print()
11
12    a, b, c = eval(input("Please enter the coefficients (a, b, c): "))
13
14    discRoot = math.sqrt(b * b - 4 * a * c)
15    root1 = (-b + discRoot) / (2 * a)
16    root2 = (-b - discRoot) / (2 * a)
17
18    print()
19    print("The solutions are:", root1, root2 )

```

Listing 2.8: Programa quadratic.py

2.5 Conversão e arredondamento entre tipos de dados

Em expressões que envolvem números inteiro (*int*) e reais (*float*) o Python converte os números inteiros para números reais. Na conversão de um número real para um inteiro perde-se informação. Pois a conversão consiste em considerar apenas a parte inteiro do número e descartar a parte fracionário do número. Exemplos:

```
1 >>> 5 * 2
2 10
3 >>> 5.0 * 2
4 10.0
5 >>> 5 / 3
6 1.6666666666666667
7 >>> int(2.6)
8 2
9 >>> float(2)
10 2.0
11 >>> int("32")
12 32
13 >>> float("32")
14 32.0
```

Listing 2.9: Python Shell. Exemplos de conversão entre tipos.

O arredondamento de um número permite controlar a quantidade informação que se perde. A função `round()` permite arredondar um número real e tem a seguinte:

```
round( x [, n] )
```

Assim, podemos converter um número real (`x`) para outro número real com menos casas decimais (`n`). Para isso basta fornecer um segundo parâmetro que especifica o número de dígitos depois do ponto decimal (ver 2.10). A listagem 2.10 apresenta alguns exemplos com as funções `floor()`, `ceil()` e `round()`.

```

1 def main():
2
3     import math
4
5     print ("round(9.45, 2) : ", round(9.45, 2))
6     print ("round(9.45, 1) : ", round(9.45, 1))
7     print ("round(9.45, 0) : ", round(9.45, 0))
8
9     print ("math.ceil(9.45) : ", math.ceil(9.45))
10    print ("math.floor(9.45) : ", math.floor(9.45))
11    print ("math.ceil(9.65) : ", math.ceil(9.65))
12    print ("math.floor(9.65) : ", math.floor(9.65))
13
14    print ("math.ceil(-9.45) : ", math.ceil(-9.45))
15    print ("math.floor(-9.45) : ", math.floor(-9.45))
16    print ("math.ceil(-9.65) : ", math.ceil(-9.65))
17    print ("math.floor(-9.65) : ", math.floor(-9.65))
18
19    # https://www.tutorialspoint.com/python/
20    print ("round(80.23456, 2) : ", round(80.23456, 2))
21    print ("round(80.235, 2) : ", round(80.235, 2))
22    print ("round(80.2351, 2) : ", round(80.2351, 2))
23    print ("round(100.000056, 3) : ", round(100.000056, 3))
24    print ("round(-100.000056, 3) : ", round(-100.000056, 3))
25
26    main()
27
28    round(9.45, 2) :   9.45
29    round(9.45, 1) :   9.4
30    round(9.45, 0) :   9.0
31    math.ceil(9.45) :  10
32    math.floor(9.45) :   9
33    math.ceil(9.65) :  10
34    math.floor(9.65) :   9
35    math.ceil(-9.45) :  -9
36    math.floor(-9.45) : -10
37    math.ceil(-9.65) :  -9
38    math.floor(-9.65) : -10
39    round(80.23456, 2) :  80.23
40    round(80.235, 2) :  80.23
41    round(80.2351, 2) :  80.24
42    round(100.000056, 3) : 100.0
43    round(-100.000056, 3) : -100.0
44

```

Listing 2.10: Python Shell. Exemplos de conversão entre tipos reais.

2.6 Acumulador

A listagem 2.11 ilustra o cálculo do valor do fatorial do número 6 ($6 \times 5 \times 4 \times 3 \times 2 \times 1 \times$) utilizando um acumulador. Ou seja, o valor obtido no passo anterior é reutilizado no passo corrente. Este algoritmo é conhecido como um acumulador, porque estamos construindo para cima ou para acumular a resposta em uma variável, conhecido como a variável acumulador.

```

1 Como calculamos 6 !?
2 6 * 5 = 30
3 Tome esse 30 e 30 * 4 = 120
4 Tome esse 120 e 120 * 3 = 360
5 Tome esse 360 e 360 * 2 = 720
6 Tome esse 720 e 720 * 1 = 720

```

Listing 2.11: Cálculo do valor de 6!

A forma geral de um algoritmo acumulador é a seguinte:

```
1 Inicializar a variável acumulador
2 Iterar até o resultado final ser alcançado
3   Atualizar o valor da variável acumulador
```

Listing 2.12: forma geral de um algoritmo acumulador

A forma geral de um algoritmo acumulador é a seguinte:

```
1 fact = 1
2 for x in [6, 5, 4, 3, 2]:
3     fact = fact * x
4 print(fact)
```

Listing 2.13: Programa para calcular 6! utilizando um acumulador.

2.7 Função range

A função `range()` permite gerar um sequência de números. A sua sintaxe tem as seguintes possibilidades:

`range(<número _inteiro>)`

`range(<número_inicial>, <número_final>)`

`range(<número_inicial>, <número_final>, <incremento>)`

A listagem 1 apresenta exemplos para as diversas possibilidades de reutilização da função `range()`.

```
1 >>> for x in range(5): print(x)
2 0
3 1
4 2
5 3
6 4
7 >>> for x in range(2, 6): print(x)
8 2
9 3
10 4
11 5
12 >>> for x in range(5, 10, 2): print(x)
13 5
14 7
15 9
16 >>> for x in range(15, 9, -3): print(x)
17 15
18 12
```

Listing 2.14: Exemplo da utilização da função `range()`.

A listagem 2.15 ilustra um programa para calcular o fatorial de um número utilizando a estrutura de repetição *for* e a função `range()` para gerar um sequência de números. A listagem 2.16 apresenta uma execução do programa para o número 100. O resultado é um número enorme.


```

1 # factorial.py
2 #   Program to compute the factorial of a number
3 #   Illustrates for loop with an accumulator
4
5 def main():
6     n = eval(input("Please enter a whole number: "))
7     fact = 1
8     for factor in range(n,1,-1):
9         fact = fact * factor
10    print("The factorial of", n, "is", fact)
11
12 main()

```

Listing 2.15: Programa para calcular o fatorial de um número utilizando for e range().

Listing 2.16: Fatorial do número 100.

```

Please enter a whole number: 100
The factorial of 100 is 933262154439441526816992388562667004
907159682643 816214685929638952175999932299156089414639761565
182862536979208272237582511852109 1686400000000000000000000000
00

```

2.8 Função list

A função `list()` permite gerar uma lista a partir de uma sequência de valores. `list(<sequence>)` to make a list

```

1 >> list(range(10))
2 [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
3 >>> list(range(1, 11))
4 [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
5 >>> list(range(0, 30, 5))
6 [0, 5, 10, 15, 20, 25]
7 >>> list(range(0, 10, 3))
8 [0, 3, 6, 9]
9 >>> list(range(0, -10, -1))
10 [0, -1, -2, -3, -4, -5, -6, -7, -8, -9]
11 >>> list(range(0))
12 []
13 >>> list(range(1, 0))
14 []

```

Listing 2.17: Exemplo da utilização da função list().

2.9 Strings

Uma string é uma sequência de caracteres entre aspas (") ou apóstrofo ('). A listagem 2.21 apresenta exemplos de strings e operações com elas. A tabela 2.6 apresenta uma string que representa um nome e a posição de cada letra do referido nome.

No endereço <http://www.openbookproject.net/thinkcs/python/english2e/ch07.html> estão disponibilizados muitos exemplos que operam com strings.

Tabela 2.6: Letras de uma string e respetiva posição.

Letra	A	n	a		M	a	r	i	a		G	o	m	e	s
Posição	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14

```

1 >>> nome = "Ana"
2 >>> apelido = 'Gomes'
3 >>> print(nome, apelido)
4 Ana Gomes
5 >>> nome_completo = nome + ' ' + apelido
6 >>> print(nome_completo)
7 Ana Gomes
8 >>>
9 >>> nome = 'Ana Maria Gomes'
10 >>> nome[0]
11 'A'
12 >>> print(nome[0], nome[4], nome[10])
13 A M G
14 >>> nome[-1]
15 's'
16 >>> nome[-2]
17 'e'
18 >>> nome[0:4]
19 'Ana '
20 >>> nome[0:3]
21 'Ana'
22 >>> nome[4:6]
23 'Ma'
24 >>> nome[4:9]
25 'Maria'
26 >>> nome[:9]
27 'Ana Maria'
28 >>> nome[9:]
29 'Gomes'
30 >>> nome[10:]
31 'Gomes'
32 >>> 3 * 'A' + 2 * 'B'
33 'AAABB'
34 >>> len(nome)
35 15
36

```

Listing 2.18: Exemplo de strings e operações.

O processamento de texto é uma das funções mais utilizada em computadores. A manipulação de texto envolve muitas operações sendo as mais comuns procurar texto e procurar e substituir texto. Outras operação envolvem a manipulação de texto. Por exemplo retirar o nome e o apelido de uma string que contem o nome completo de uma pessoa: Exemplo: dado o nome o completo: "Ana Pereira Gomes", obter o nome: "Ana" e o apelido; "Gomes". Na tabela 2.7 são apresentados alguns operadores para strings incluindo exemplos.

Podemos ter acesso aos caracteres individuais em uma string através de indexação `[]`. As posições em uma string são numeradas a partir da esquerda, começando com 0. A forma geral é `<string> [<expr>]`, onde o valor de `expr` determina qual caractere é selecionado a partir da cadeia.

2.9.1 Padrão travessia

Muitos cálculos em computador envolvem o processamento de uma sequência de caracteres, sendo processado um de cada vez. Muitas vezes, os algoritmos desenvolvidos para este tipo de processamento começam por selecionar o primeiro elemento, fazem um ou mais operações com ele e passam ao próximo elemento. Este processo repete-se até se atingir o último elemento. Este padrão de processamento é chamado de travessia.

Tabela 2.7: Operadores para manipulação de strings

Operador	Significado
+	Concatenação
*	Repetição
<string>[]	indexação
<string>[<start>:<end>]	Sub-strings, não inclui o caractere na posição <end>
len(<string>)	Comprimento
for <var> in <string>	Iteração através de personagens

Uma maneira de codificar uma travessia é com estruturas de repetição (*while*, *for*). Na listagem 2.19 apresentado um exemplo utilizando o *for* em na listagem 2.20 é apresentado o mesmo exemplo com o *while* [?].

```

1 nome = 'Ana Maria Gomes '
2 index = 0
3 while index < len(nome):
4     letter = nome[index]
5     print (letter)
6     index += 1

```

Listing 2.19: Travassia com while.

O ciclo *while* percorre a string e exibe cada letra numa linha. A condição de ciclo é o `index < len (nome)`, de modo que quando o `index` é igual ao comprimento da string, a condição é falsa, e o corpo do ciclo não é executado. O último caractere acedido é aquele com o índice `len (nome) - 1`, que é o último caractere na string.

Usando um índice para percorrer um conjunto de valores é tão comum que Python oferece uma sintaxe alternativa simplificada, apresentada na listagem 2.20. A variável `c` (na linha 2) vai assumir cada uma dos caracteres da variável `nome` que é imprimido com a instrução `print(c)` (linha 3).

```

1 nome = 'Ana Maria Gomes '
2 for c in nome:
3     print (c)

```

Listing 2.20: Travessia com for.

Listing 2.21: Saída dos programas travessia com while e for .

A
n
a

M
a
r
i
a

G
o
m
e
s

A listagem 2.22 apresenta um programa que dado o número do mês apresenta as iniciais em inglês do respectivo mês. A variável **months** contém as iniciais (3 letras) de cada mês umas a seguir às outras começando pelo mês de janeiro e terminando no mês de dezembro. A tabela 2.8 apresenta a estrutura da variável **months**. Assim, as iniciais do mês de janeiro começam na posição 0, as do mês de fevereiro na posição 3, as do mês de março na posição 6, etc. Ou seja, as iniciais de cada mês estão a uma distância de 3 das do mês anterior.

Tabela 2.8: Iniciais dos meses do ano numa string.

Letra	J	a	n	F	e	b	M	a	r	A	p	r	M	a	y	J	u	n	J	u	l	A	u	g	S	e	p	O	c	t	N	o	v	D	e	c
Posição	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35

```

1 # month.py
2 # A program to print the abbreviation of a month, given its number
3
4 def main():
5
6     # months is used as a lookup table
7     months = "JanFebMarAprMayJunJulAugSepOctNovDec"
8
9     n = int(input("Enter a month number (1-12): "))
10
11     # compute starting position of month n in months
12     pos = (n-1) * 3
13
14     # Grab the appropriate slice from months
15     monthAbbrev = months[pos:pos+3]
16
17     # print the result
18     print("The month abbreviation is", monthAbbrev + ".")

```

Listing 2.22: Travessia com for.

2.10 Objectos

Objects are Python's abstraction for data. All data in a Python program is represented by objects or by relations between objects. (In a sense, and in conformance to Von Neumann's model of a stored program computer, code is also represented by objects.)

2.11 Estruturas de repetição

A listagem 2.23 representa um programa para calcular o valor final de um investimento, dado o valor inicial, a taxa de juro e o número de anos.

```

1 # futval.py
2 #     A program to compute the value of an investment
3 #     carried n years into the future
4
5 def main():
6     print("This program calculates the future value of a 10-year investment.")
7
8     principal = eval(input("Enter the initial principal: "))
9     apr = eval(input("Enter the annual interest rate (0-1): "))
10    n = eval(input("Enter the number of years: "))
11
12    for i in range(n):
13        principal = principal * (1 + apr)
14
15    print("The value in ", n, " years is: ", principal)
16
17 main()

```

Listing 2.23: Programa futval.py

2.12 Módulos

Listing 2.24: Exemplos de definição de constantes.

```

sdsd
sds
dsd

```

2.13 A estrutura e sintaxe do Python

2.14 Definição de constantes

As constantes pode ser definidas com a função bool `define (string \ $name , mixed \ $value [, bool \ $scope])`. Por convenção, o nomes de constantes são sempre em maiúsculas e o seu escopo é global. Logo, estão disponíveis em qualquer lugar do script Python. As listagens seguintes ilustram alguns exemplos.

```

1  define("CONSTANTE", "Olá Mundo.");
2  echo CONSTANTE; // mostra "Olá Mundo."
3  echo Constante; // mostra "Constante" e dá um aviso.
4
5  define("GREETING", "Olá Você.", TRUE);
6  echo GREETING; // mostra "Olá você."
7  echo Greeting; // mostra "Olá você."
8
9  // Nomes de constantes válidos
10 define("FOO", "alguma coisa");
11 define("FOO2", "alguma outra coisa");
12 define("FOO_BAR", "alguma coisa mais");
13
14 // Nomes de constantes inválidas

```

```

15  define("2FOO",      "alguma coisa");
16
17  // Configurable, but constant for this installation
18  define('DATABASE', 'mydb');
19  define('IMAGES_DIRECTORY', '/tmp/images');

```

Listing 2.25: Exemplos de definição de constantes.

```

1  define('MIN_VALUE', '0.0'); // RIGHT - Works OUTSIDE of a class definition.
2  define('MAX_VALUE', '1.0'); // RIGHT - Works OUTSIDE of a class definition.
3
4  //const MIN_VALUE = 0.0;      WRONG - Works INSIDE of a class definition.
5  //const MAX_VALUE = 1.0;      WRONG - Works INSIDE of a class definition.
6
7  class Constants
8  {
9      //define('MIN_VALUE', '0.0'); WRONG - Works OUTSIDE of a class definition.
10     //define('MAX_VALUE', '1.0'); WRONG - Works OUTSIDE of a class definition.
11
12     const MIN_VALUE = 0.0;      // RIGHT - Works INSIDE of a class definition.
13     const MAX_VALUE = 1.0;      // RIGHT - Works INSIDE of a class definition.
14
15     public static function getMinValue()
16     {
17         return self::MIN_VALUE;
18     }
19     public static function getMaxValue()
20     {
21         return self::MAX_VALUE;
22     }
23 }

```

Listing 2.26: Exemplos de definição de constantes em classes.

2.15 Datas e horas

O website <http://lh.2xlibre.net/> é mantida uma lista de locais com os formatos para datas, horas, moeda, etc.

A listagem 2.27 mostra vários exemplos de formatação da data e hora obtidas do sistema operativo.

```

1  <?Python
2  // Assuming today is March 10th, 2001, 5:16:18 pm, and that we are in the
3  // Mountain Standard Time (MST) Time Zone
4
5  date_default_timezone_set('Europe/Lisbon');
6  $today = date("F j, Y, g:i a"); // March 10, 2001, 5:16 pm
7  $today = date("m.d.y"); // 03.10.01
8  $today = date("j, n, Y"); // 10, 3, 2001
9  $today = date("Ymd"); // 20010310
10 $today = date('h-i-s, j-m-y, it is w Day'); // 05-16-18, 10-03-01, 1631 1618 6 Satpm01
11 $today = date('\i\t \i\s \t\h\e jS \d\a\y.'); // it is the 10th day.
12 $today = date("D M j G:i:s T Y"); // Sat Mar 10 17:16:18 MST 2001
13 $today = date('H:m:s \m \i\s\ \m\o\n\t\h'); // 17:03:18 m is month
14 $today = date("H:i:s"); // 17:16:18
15 $today = date("Y-m-d H:i:s"); // 2001-03-10 17:16:18 (the MySQL DATETIME
    format)

```

Listing 2.27: Exemplos de formatação de data e horas.

2.15.1 Número de dias até uma data

The example below outputs the number of days until 4th of July:

```
1 <?Python
2 $d1=strtotime("July 04");
3 $d2=ceil(($d1-time())/60/60/24);
4 echo "There are " . $d2 . " days until 4th of July.";
```

Listing 2.28: Número de dias até 4 de julho.

Capítulo 3

Operações com a Internet

3.1 Obter páginas web com *urlopen*

O script Python da listagem 3.1 permite obter o números de alunos inscritos por curso no IPG. O parâmetro $P_{CURSIGLA} : 'LCM'$ indica o curso. Neste caso o Curso de Licenciatura de Comunicação Multimédia.

```
1 #import requests
2 import urllib.request
3 import os
4 import sys
5 import re
6 import zipfile
7
8 import urllib.parse
9 import urllib.request
10
11 try:
12     # For Python 3.0 and later
13     from urllib.request import urlopen
14 except ImportError:
15     # Fall back to Python 2's urllib2
16     from urllib2 import urlopen
17
18 PLUGINLIST = "urls.txt"
19 cursos = open(PLUGINLIST, "r").readlines()
20
21 values = {
22     'P_COD' : '',
23     'P_ANO' : '',
24     'P_MODAL' : '1',
25     'P_ANO_UPP' : '',
26     'P_CUR_SIGLA' : 'LCM',
27     'P_ESTADO' : 'F',
28     'P_N_REGISTOS' : '20',
29     'P_WEB_PAGE' : '',
30     'P_NOME' : '',
31     'P_RAMO' : '',
32     'P_EMAIL' : '',
33     'P_sort' : '1',
34     'p_tipo' : '',
35     'P_TF' : '',
36     'P_start' : '1'}
37
38 url = 'https://cloud.sysnovare.pt/ipg/alunos_geral.QueryList '
39 # for curso in cursos:
40     # curso = curso[0:len(curso)-1]
41     # html = urlopen(url)
42     # print(html.read())
43     # values ['P_CUR_SIGLA'] = curso
44     # data = urllib.parse.urlencode(values)
45     # data = data.encode('ascii') # data should be bytes
46     # req = urllib.request.Request(url, data)
```

```

47 # with urllib.request.urlopen(req) as response:
48 #     the_page = response.read()
49 # #print (the_page)
50 # #Registos: 41 a 60 de um total de 114
51
52 # the_page = str(the_page)
53 # #matches = re.findall('(\\d+)', the_page)
54 # matches = re.findall('(um total de\\s+)(\\d+)', the_page)
55 # print (curso, matches[0][1])
56 tc = ['TeSP','L','M','E']
57 print (tc)
58 nt = 0
59 for t in tc:
60     url2 = 'https://cloud.sysnovare.pt/ipg/cursos_geral.apresentacao?P_grau=' + t
61     #https://cloud.sysnovare.pt/ipg/cursos_geral.FormView?P_CUR_SIGLA=LASC
62     req = urllib.request.Request(url2)
63     with urllib.request.urlopen(req) as response:
64         the_page = response.read()
65         the_page = str(the_page)
66         matches = re.findall('(P_CUR_SIGLA=)(\\w+)', the_page)
67         for curso in matches:
68             curso = curso[1]
69             html = urlopen(url)
70             #print (curso)
71             #print(html.read())
72             values ['P_CUR_SIGLA'] = curso
73             data = urllib.parse.urlencode(values)
74             data = data.encode('ascii') # data should be bytes
75             req = urllib.request.Request(url, data)
76             n = 0
77             try:
78                 with urllib.request.urlopen(req) as response:
79                     the_page = response.read()
80                     #Registos: 41 a 60 de um total de 114
81                     the_page = str(the_page)
82                     matches2 = re.findall('(um total de\\s+)(\\d+)', the_page)
83                     if (len(matches2)>0):
84                         n = matches2[0][1]
85             except urllib.error.URLError as e:
86                 print(e.reason)
87             print (curso, n)
88             a=0
89             try:
90                 a = eval(str(n))
91             except (ValueError, SyntaxError):
92                 pass
93             nt = nt + a
94         print ('Total', t, nt)
95
96 #LOC = eval(endline) - eval(beginline) + 1

```

O script Python da listagem 3.1 ilustra a utilização da função *urlencode* pra codificar parâmetros URL de pedidos HTTP.

```

1
2 import urllib.request
3 import urllib.parse
4 data = {}
5 data['name'] = 'Somebody Here'
6 data['location'] = 'Northampton'
7 data['language'] = 'Python'
8 url_values = urllib.parse.urlencode(data)
9 print(url_values) # The order may differ from below.
10 name=Somebody+Here&language=Python&location=Northampton
11 url = 'http://www.example.com/example.cgi'
12 full_url = url + '?' + url_values
13 data = urllib.request.urlopen(full_url)

```

Listing 3.1: Codificação de parâmetros (*urlencode*)

Capítulo 4

Cálculo de perímetros, áreas e volumes

4.1 Perímetro de um retângulo

Elabore um algoritmo/programa que permita calcular o perímetro de um retângulo.

4.1.1 Desenvolvimento do algoritmo

4.1.1.1 Modelo

A figura ?? ilustra o modelo para calcular o perímetro. A formula seguinte permite calcular o seu valor:

$$P = 2 \times L + 2 \times C$$

onde:

- L - Largura do retângulo ($L \in \mathbb{R}$)
- C - Comprimento do retângulo ($C \in \mathbb{R}$)
- P - Perímetro do retângulo ($P \in \mathbb{R}$)

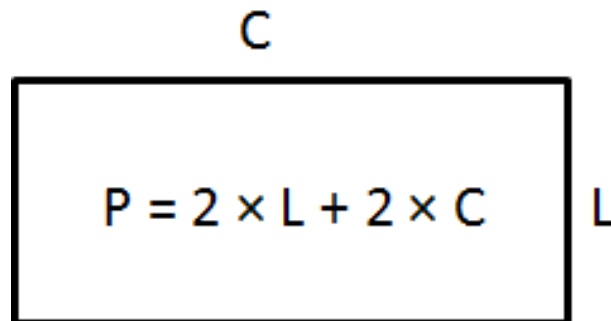


Figura 4.1: Modelo perímetro de um retângulo.

4.1.1.2 Esboço

Listing 4.1: Esboço do algoritmo perímetro de um retângulo.

```

LER C
LER L
 $P = 2 \times C + 2 \times L$ 
ESCREVER P

```

4.1.1.3 Algoritmo

Listing 4.2: Algoritmo perímetro de um retângulo sem validação de dados de entrada.

```

Algoritmo: algoritmoPerimetroRetangulo
Objetivo: Permite calcular o perímetro de um retângulo
Variáveis
Entrada:
    L (Inteiro T2) – Largura (> 0, <= 99)
    C (Inteiro T2) – Comprimento (> 0, <= 99)
Saída:
    P (Inteiro T5) – Perímetro (> 0, <= 99999)
Data: 2016-9-26 13:46:11
Autor: Paulo Nunes
Versão: 1.0
Obs:
Início:
    /* Entrada de dados (INPUT) */
    ESCREVER "Largura?"
    ESCREVER "Comprimento?"
    /* Processamento (PROCESSING) */
     $P = 2 \times L + 2 \times C$ 
    /* Saída de resultados (OUTPUT) */
    ESCREVER "Perímetro: ", P
Fim.

```

Listing 4.3: Algoritmo perímetro de um retângulo.

```

Algoritmo: algoritmoPerimetroRetangulo
Objetivo: Permite calcular o perímetro de um retângulo
Variáveis
Entrada:
    L (Inteiro T2) – Largura (> 0, <= 99)
    C (Inteiro T2) – Comprimento (> 0, <= 99)
Saída:
    P (Inteiro T5) – Perímetro (> 0, <= 99999)
Data: 2016-9-26 13:46:11
Autor: Paulo Nunes
Versão: 1.0
Obs:
Início:
/* Entrada de dados (INPUT) */
FAZER
    ESCREVER "Largura?"
    LER L
ATÉ ( (L > 0) E (L <= 99) )
FAZER
    ESCREVER "Comprimento?"
    LER C
ATÉ ( (C > 0) E (C <= 99) )
/* Processamento (PROCESSING) */
P = 2 x L + 2 x C
/* Saída de resultados (OUTPUT) */
ESCREVER "Perímetro: ", P
Fim.

```

4.1.2 Programa

```

1 def main():
2     L = eval(input("Largura do retângulo: "))
3     C = eval(input("Comprimento do retângulo: "))
4     P = 2 * L + 2 * C
5     print("Perímetro do retângulo ", P)
6
7 main()

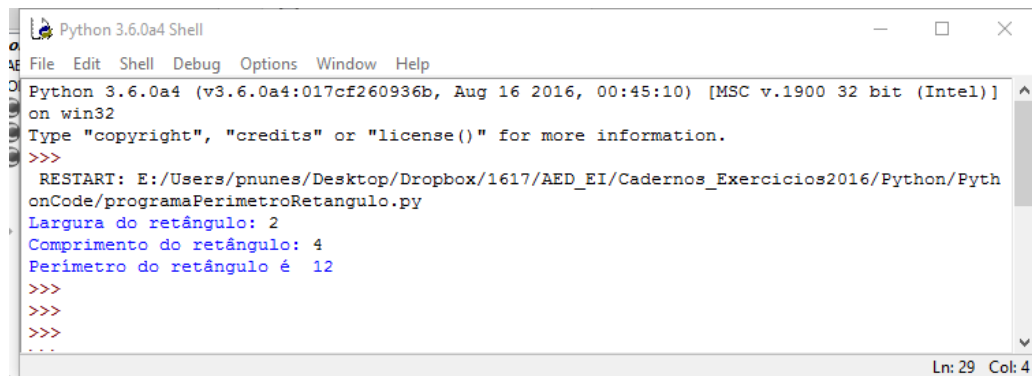
```

Listing 4.4: Programa Python perímetro de um retângulo

4.1.2.1 Caso 1

4.2 Volume paralelepípedo

Elabore um algoritmo/programa que permita calcular o volume de um paralelepípedo, de acordo com a figura seguinte (4.3).



```

Python 3.6.0a4 Shell
File Edit Shell Debug Options Window Help
Python 3.6.0a4 (v3.6.0a4:017cf260936b, Aug 16 2016, 00:45:10) [MSC v.1900 32 bit (Intel)]
on win32
Type "copyright", "credits" or "license()" for more information.
>>>
RESTART: E:/Users/pnunes/Desktop/Dropbox/1617/AED_EI/Cadernos_Exercicios2016/Python/Pyth
onCode/programaPerimetroRetangulo.py
Largura do retângulo: 2
Comprimento do retângulo: 4
Perímetro do retângulo é 12
>>>
>>>
>>>
...
Ln: 29 Col: 4

```

Figura 4.2: Caso 1: perímetro de um retângulo.

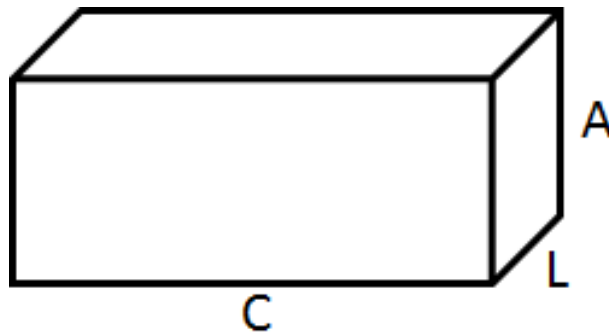


Figura 4.3: Modelo perímetro de um retângulo.

4.2.1 Desenvolvimento do algoritmo

4.2.1.1 Modelo

A figura ?? ilustra o modelo para calcular o volume de um paralelepípedo. A formula seguinte permite calcular o seu valor:

$$V = C \times L + \times C \times A$$

onde:

- L - Largura do paralelepípedo ($L \in \mathbb{R}$)
- C - Comprimento do paralelepípedo ($C \in \mathbb{R}$)
- A - Altura do paralelepípedo ($A \in \mathbb{R}$)
- V - Volume do paralelepípedo ($V \in \mathbb{R}$)

4.2.1.2 Esboço

Listing 4.5: Esboço do algoritmo o volume de um paralelepípedo.

```

LER C
LER L
LER A
 $V = C \times L \times A$ 
ESCREVER V

```

4.2.1.3 Algoritmo

Listing 4.6: Algoritmo Programa Python volume de um paralelepípedo

Algoritmo: programaVolumeParalelepipedo

Objetivo:

Permite calcular o volume de um paralelepípedo

Variáveis

Entrada:

L (Real T6.3) – Largura do paralelepípedo (m) (> 0 , ≤ 999.999)
 C (Real T6.3) – Comprimento do paralelepípedo (m) (> 0 , ≤ 999.999)
 A (Inteiro T6.3) – Altura do paralelepípedo (m) (> 0 , ≤ 999.999)

Saída:

V (Real T12.3) – Volume do paralelepípedo (m³) (> 0.0 , ≤ 999999999)

Data: 2016-9-26 19:4:15

Autor: Paulo Nunes

Versão: 1.0

Obs:

Início:

```

/* Entrada de dados (INPUT) */
FAZER
    ESCREVER "Largura do paralelepípedo (m)?"
    LER L
ATÉ ( (L > 0) E (L <= 999.999) )
FAZER
    ESCREVER "Comprimento do paralelepípedo (m)?"
    LER C
ATÉ ( (C > 0) E (C <= 999.999) )
FAZER
    ESCREVER "Altura do paralelepípedo (m)?"
    LER A
ATÉ ( (A > 0) E (A <= 999.999) )
/* Processamento (PROCESSING) */
 $V = C * L * A$ 
/* Saída de resultados (OUTPUT) */
ESCREVER "Volume do paralelepípedo: ", V, " m3"

```

Fim.

4.2.1.4 Programa

```

1 def programaVolumeParalelepipedo():
2     L = -1
3     while ((L <= 0.0) or (L >= 999.999)):
4         L = eval(input("Largura do paralelepípedo (m)? "))
5     C = -1
6     while ((C <= 0.0) or (C >= 999.999)):
7         C = eval(input("Comprimento Largura do paralelepípedo (m)? "))
8     A = -1
9     while ((A <= 0.0) or (A >= 999.999)):
10        A = eval(input("Altura Largura do paralelepípedo (m)? "))
11    V = C * L * A
12    print("Volume do paralelepípedo: ", V, " m3")
13
14 programaVolumeParalelepipedo()

```

Listing 4.7: Programa Python volume de um paralelepípedo

4.2.1.5 Caso 1

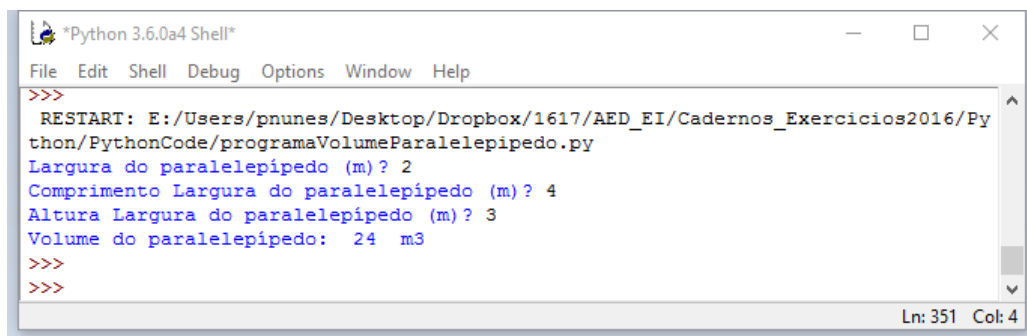


Figura 4.4: Caso 1: Programa Python volume de um paralelepípedo.

4.3 Proposta de exercícios

4.3.1 Perímetro de uma rotunda

Elabore um algoritmo para calcular o perímetro de uma rotunda. Assim como o número peças de cimento para circundar a rotunda.

4.3.2 Área de uma rotunda

Elabore um algoritmo para calcular a área de uma rotunda. Assim como o número de pés de flor para plantar na rotunda.

4.3.3 Volume de um cilindro

Elabore um algoritmo para calcular o volume de um cilindro.

Capítulo 5

Cálculo de médias e classificações qualitativas

5.1 Média de três números

Elabore um algoritmo para calcular a média de três número.

5.2 Média de vários números

Elabore um algoritmo para calcular a média das notas dos alunos de uma disciplina.

5.3 Classificação qualitativa

Elabore um algoritmo em fluxograma para atribuir uma classificação qualitativa de acordo com a seguinte tabela:

Tabela 5.1: Tabela de classificação

Nota	Classificação
0 a 9	Aprovado
10 a 20	Reprovado

5.4 Qualidade do ar

Elabore um algoritmo que permita determinar a qualidade do ar com base na concentração de CO.

Poluente em causa / Classificação	CO	
	Min	Máx
Mau	10000	-----
Fraco	8500	9999
Médio	7000	8499
Bom	5000	6999
Muito Bom	0	4999

Figura 5.1: Classificação do Índice de Qualidade do Ar proposto para o ano 2013 ($\mu\text{g}/\text{m}^3$)

Capítulo 6

Operações com data e horas

A listagem 6.1 mostra vários exemplos de formatação da data e hora obtidas do sistema operativo.

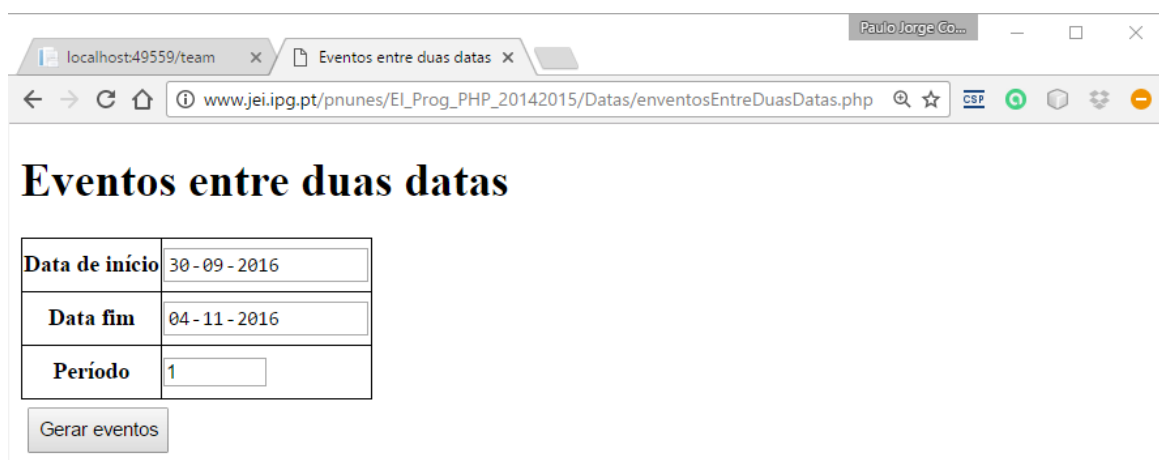
Listing 6.1: Exemplos de formatação de data e horas.

...

6.1 Contagem do número de dias de um dado dia da semana entre duas datas

Elabore um algoritmo que permita calcular o número de dias de um dado dia da semana (seg, ter, ..., dom) entre duas datas. Exemplo:

6.2 Contagem do ocorrências entre duas datas com periodicidade



The screenshot shows a web browser window with the address bar displaying 'localhost:49559/team' and 'Eventos entre duas datas'. The page title is 'Eventos entre duas datas'. The form contains the following fields:

Data de início	30-09-2016
Data fim	04-11-2016
Período	1

Below the form is a button labeled 'Gerar eventos'.

Figura 6.1: Interface eventos entre duas datas com periodicidade

Eventos entre 2016-09-19 e 2016-12-19

Período 7

```
1 - 2016-09-19
2 - 2016-09-26
3 - 2016-10-03
4 - 2016-10-10
5 - 2016-10-17
6 - 2016-10-24
7 - 2016-10-31
8 - 2016-11-07
9 - 2016-11-14
10 - 2016-11-21
11 - 2016-11-28
12 - 2016-12-05
13 - 2016-12-12
14 - 2016-12-19
```

6.3 Número de dias até uma data

...

6.3.1 Geração de datas de eventos recorrentes

As agendas eletrônicas permitem gerar uma sequência de datas de eventos recorrentes. Com base na data do primeiro evento, o número de eventos e o período de tempo que cada evento se repete (semana, quinzena) geram a sequência de datas dos eventos. Neste exercício pretende-se gerar uma sequência de datas de eventos entre duas datas com uma dada periodicidade fornecidos pelo utilizador.

A figura 6.2 ilustra um exemplo de geração de eventos com um período de 7 dias.

(a) Input

(b) Output

Figura 6.2: Eventos entre duas datas.

Listing 6.2: ...

...

6.3.2 Eventos em tabela com nome do mês do dia da semana

Altere o exemplo anterior de forma a apresentar o resultado numa tabela HTML com o aspeto da figura 6.3. Utilize o código da listagem 6.3 para seleccionar zona de tempo Europe/Lisbon e local *pt_BR*.

Listing 6.3: Seleção de zona de tempo (Europe/Lisboa) e localidade (Portugal: pt_BR).

A figura ?? ilustra um exemplo de geração de eventos com um período de 7 dias.

(a) Input

(b) Output

Figura 6.3: Eventos entre duas datas numa tabela HTML em português.

6.3.3 Eventos em tabela na língua inglesa

Altere o exemplo anterior de forma a apresentar o resultado na língua inglesa com o aspeto da figura 6.4. Para a resolução deste exercício deve seleccionar a língua inglesa no início do script utilizando a função. A listagem 6.4 ilustra um exemplo de seleção da língua portuguesa e da língua.

Listing 6.4: Seleção de zona de tempo (Europe/London) e localidade (en_GB).

```
<?php
date_default_timezone_set( 'Europe/London' );
setlocale( LC_ALL, 'en_GB' );
// ...
```

A figura ?? ilustra um exemplo de geração de eventos com um período de 7 dias.



(a) Input



(b) Output

Figura 6.4: Eventos entre duas datas numa tabela HTML em inglês.

6.3.3.1 Eventos em tabela numa dada língua

Adicione uma caixa de seleção ao interface da figura ?? de modo permitir ao utilizador seleccionar a língua na qual deseja obter o output. Efetue as alterações necessárias no código que gera o output para produzir o output de acordo com a língua seleccionada.

6.3.3.2 Eventos tabela HTML com interrupções

Considere o exercício da secção 6.3.3.1. Adicione uma tabela com duas caixas de entradas de dados, data de início e data de fim de interrupção, um botão para dicionar linhas no fim da tabela e um botão para remover a última linha da tabela. A figura ?? ilustra o aspecto do interface pretendido.

A figura ?? mostra o output dos eventos com início a 25-02-2015 e fim a 19-05-201, uma periodicidade de 7 dias e com duas interrupções. A primeira, para o Carnaval, entre os dias 16-02-2015 a 17-02-2015 e a segunda, para a Páscoa, entre os dias 30-03-2015 a 06-04-2015.

Capítulo 7

Operações com números

7.1 Decompor os dígitos de um número

Elabore um programa para decompor os dígitos de um número.

Capítulo 8

Ciclos de repetição

8.1 Ciclo while

A while loop statement in Python programming language repeatedly executes a target statement as long as a given condition is true.

A sintaxe do ciclo while é:

```
while expression:
    statement(s)
```

```
1 contador = 0
2 while (contador <= 9):
3     print ('O contador Ã©:', contador)
4     contador = contador + 1
```

Listing 8.1: Programa contador com ciclo while

Listing 8.2: Resultado do programa contador com ciclo while

```
O contador é: 0
O contador é: 1
O contador é: 2
O contador é: 3
O contador é: 4
O contador é: 5
O contador é: 6
O contador é: 7
O contador é: 8
O contador é: 9
>>>
```


Capítulo 9

Estruturas de dados

http://www.thomas-cokelaer.info/tutorials/python/data_structures.html

binary files: <https://docs.python.org/2/tutorial/inputoutput.html>

tutorial: <https://docs.python.org/3/library/io.html>

9.1 7.3. struct ? Interpret strings as packed binary data

<https://docs.python.org/2/library/struct.html?highlight=struct#module-struct>

Capítulo 10

Dicionário em ficheiro

<https://docs.python.org/3/library/shelve.html>

Capítulo 11

Tipos de dados

11.1 Calendários datas e horas

`datetime` – Basic date and time types:

<https://docs.python.org/3/library/index.html>

<http://www.cyberciti.biz/faq/howto-get-current-date-time-in-python/>

```

1 import time
2
3 now = time.strftime("%c")
4 ## date and time representation
5 print "Current date & time " + time.strftime("%c")
6
7 ## Only date representation
8 print "Current date " + time.strftime("%x")
9
10 ## Only time representation
11 print "Current time " + time.strftime("%X")
12
13 ## Display current date and time from now variable
14 print ("Current time %s" % now )
15
16
17
18 import datetime
19 i = datetime.datetime.now()
20
21 print ("Current date & time = %s" % i)
22
23 print ("Date and time in ISO format = %s" % i.isoformat() )
24
25 print ("Current year = %s" %i.year)
26
27 print ("Current month = %s" %i.month)
28
29 print ("Current date (day) = %s" %i.day)
30
31 print ("dd/mm/yyyy format = %s/%s/%s" % (i.day, i.month, i.year) )
32
33 print ("Current hour = %s" %i.hour)
34
35 print ("Current minute = %s" %i.minute)
36
37 print ("Current second = %s" %i.second)
38
39 print ("hh:mm:ss format = %s:%s:%s" % (i.hour, i.month, i.second) )
40
41
42
43 import time
44
45 start = start = time.time()
46 with open("test.txt", 'w') as f:
47     for i in range(10000000):
48         # print('This is a speed test', file=f)
49         # f.write('This is a speed test\n')
50 end = time.time()
51 print(end - start)

```

11.2 Calendar - General calendar-related functions

<https://docs.python.org/3/library/calendar.html>