1. Computadores, Algoritmos e Programas

1.1. Características de um computador

→ Um computador é uma máquina cuja função é manipular símbolos.

→ É automático, pois não necessita da intervenção humana para a resolução de um problema. O computador recebe um conjunto de instruções (programa) de modo a chegar a uma solução. Estas instruções são escritas numa notação compreendida pelo computador (linguagem de programação) que especificam como a tarefa deve ser executada.

→ É universal, porque pode efetuar qualquer tarefa cuja solução possa ser expressa através de um programa. "universal" provém do facto de o computador poder executar qualquer programa, resolvendo problemas em diferentes áreas de aplicação.

→ De acordo com Church-Turing, qualquer computação pode ser baseada num pequeno número de operações elementares:

- Operações de entrada de dados (obtêm valores exterior do programa);
- Operações de saída de dados (mostram valores existentes no programa);
- Operações matemáticas (efetuam cálculos sobres os dados existentes no programa);
- Execução condicional (teste de certas condições e execução de instruções, ou não, dependendo do resultado do teste;
- Repetição (execução repetitiva de certas instruções).

→ A tarefa de programação corresponde a dividir um problema grande e complexo em vários problemas, até se atingirem operações elementares.

→ É eletrónico.

→ É digital, já que efetua operações sobre informação que é codificada recorrendo a duas grandezas discretas (0 e 1) e não sobre grandezas que variam de um modo contínuo.

1.2. Algoritmos

Um programa segue um conjunto de instruções bem definidas que especificam exatamente o que tem que ser feito. Este conjunto de instruções é caracterizado matematicamente como um algoritmo. Um programa, assim, corresponde a um

algoritmo escrito numa linguagem que é entendida pelo computador, chamada de linguagem de programação.

Definição: Um algoritmo é uma sequência finita de instruções (existe uma ordem pela qual as instruções aparecem no algoritmo e são em número finito) bem definidas e não ambíguas (instruções têm significado claro, não havendo lugar para múltiplas interpretações), cada uma das quais pode ser executada mecanicamente num período de tempo finito com uma quantidade de esforço finita.

Um algoritmo está sempre associado à solução de um dado problema. A execução das instruções do algoritmo garante que o seu objetivo é atingido.

1.2.2. Características de um algoritmo

- → É rigoroso, pois deve especificar rigorosamente o que deve ser feito, não havendo lugar para ambiguidade. Nota-se que para evitar a ambiguidade inerente à linguagem natural, criaram-se novas linguagens – linguagens artificiais – de modo a exprimir os algoritmos de um modo rigoroso.
- → É eficaz, já que cada instrução do algoritmo deve ser suficientemente básica e bem compreendida de modo a poder ser executada num intervalo de tempo finito, com uma quantidade de esforço finita.
- → Deve terminar, ou seja, um algoritmo deve levar a uma situação em que o objetivo que tenha sido atingido e não existam mais instruções para executadas.

1.3. Programas e algoritmos

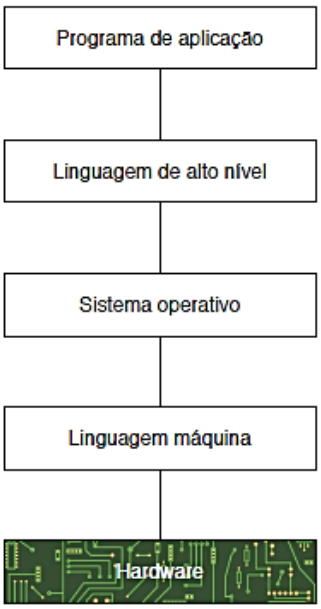
Um algoritmo, escrito de modo a poder ser executado por um computador, tem o nome de programa.

Tipicamente, o computador começa por receber certos valores para algumas das variáveis, depois efetua operações sobre essas variáveis, possivelmente atribuindo valores a novas variáveis e, finalmente, chega a um conjunto de valores que constituem o resultado do programa. Estas ações devem ser suficientemente elementares, ou seja, devem ser um conjunto de ações simples a serem executadas numa sequência bem definida, para poderem ser executadas facilmente pelo agente que executa o algoritmo.

1.3.1. Linguagens de programação

Uma linguagem de programação é utilizada para escrever programas de computador, podem ser classificadas em linguagens máquina, linguagens "assembly" e linguagens de alto nível.

- A linguagem máquina é a utilizada para comandar diretamente as ações do computador. São constituídas por sequencias de dois símbolos discretos, correspondendo à existência (1) ou à ausência de sinal (0) e manipulam diretamente entidades dentro do computador. É difícil de usar, de compreender e varia de computador para computador.
- A linguagem "assembly" é semelhante à linguagem máquina, diferindo desta no sentido em que nomes simbólicos com significado para humanos em ligar de sequências de zeros e uns.
- As linguagens de alto nível aproximam-se da linguagem natura, para além de poderem ser utilizadas em computadores diferentes.



Para que os computadores possam "entender" os programas escritos numa linguagem de alto nível existem programas que traduzem as instruções de linguagens de alto nível em linguagem máquina, chamados processadores da linguagem. Há dois processos para fazer esta tradução: por compilação e por interpretação.

O Python, linguagem de alto nível, utiliza como tradutor um programa chamado interpretador, que recebe instruções em Python e que é capaz de executar as ações correspondentes a cada uma delas.

1.4. Sintaxe e semântica

O Python apresenta dois aspetos distintos: sintaxe e semântica da linguagem. A sintaxe determina qual a constituição das frases que podem ser fornecidas ao computador e a semântica vai determinar o que computador vai fazer ao seguir cada uma dessas frases.

1.4.1. Sintaxe

A sintaxe é o conjunto de regras que definem quais as relações válidas entre os componentes da linguagem. A sintaxe nada diz em relação ao significado das frases da linguagem.

Como a sintaxe apenas se preocupa com o processo de combinação dos símbolos de uma linguagem é facilmente formalizada.

Para descrever a sintaxe do Python, utiliza-se uma notação conhecida por notação BNF, que utiliza as seguintes regras:

- → Para designar um componente da linguagem escrevemo-lo entre parênteses angulares (**< >**). Os nomes que aparecem entre <> são designados por símbolos não terminais. Um símbolo não terminal está sempre associado a um conjunto de entidades da linguagem.
- → Os símbolos que aparecem nas frases da linguagem são chamados símbolos terminais e são escritos sem qualquer símbolo especial à sua volta.
- → O símbolo **|** (ou) representa possíveis alternativas.
- → O símbolo **::=** (é definido como) serve para definir componentes da linguagem.
- → A utilização do carácter **+** imediatamente após um símbolo não terminal significa que esse símbolo pode ser repetido uma ou mais vezes.
- → A utilização do carácter ***** imediatamente após um símbolo não terminal significa que esse símbolo pode ser repetido zero ou mais vezes.
- → A utilização de chavetas **{ }** englobando símbolos terminais ou não terminais, significa que esses símbolos são opcionais.

Nas linguagens de programação, a ambiguidade sintática não pode existir.

A notação utilizada para definir formalmente uma linguagem, no caso da notação BNF, < >, |, ::=, { }, +, *, os símbolos não terminais e os símbolos terminais, é denominada metalinguagem.

1.4.2. Semântica

A semântica de uma linguagem define qual o significado de cada frase da linguagem.

Cada frase em Python tem uma semântica, a qual corresponde às ações tomadas pelo Python ao executar essa frase. Esta semântica é definida por regras para extrair o significado de cada frase.

1.4.3. Tipos de erros num programa

Um programa pode apresentar dois tipos distintos de erros: erros de natureza sintática e erros de natureza semântica.

→ Os **erros sintáticos** resultam do facto de o programador não ter escrito as frases do seu programa de acordo com as regras da gramática da linguagem de programação utilizada. A deteção destes erros é feita pelo processador da linguagem, o qual fornece normalmente um diagnóstico sobre o que provavelmente está errado.

→ Os **erros semânticos** (erros de lógica) são erros em geral muito mais difíceis de detetar do que os erros de caráter sintático. Resultam do facto de o programador não ter expressado corretamente, através da linguagem de programação, as ações a serem executadas. Podem manifestar-se pela geração de uma mensagem de erro durante a execução de um programa, pela produção de resultados errados ou pela geração de ciclos que nunca terminam.

Ao processo de deteção e correção de erros dá-se o nome de depuração (*debugging*).

Para desenvolver programas, são necessárias duas competências fundamentais, a capacidade de resolução de problemas e a capacidade de depuração.

2. Elementos básicos de programação

2.1. Expressões

Uma expressão é entidade computacional que tem um valor.

Uma expressão em Python pode ser:

→ Uma **constante**;

Podem ser números, valores lógicos ou cadeias de caracteres. O valor de uma constante é a própria constante. A representação externa de uma entidade corresponde ao modo como nós visualizamos essa entidade, independentemente do modo como esta é representada internamente no computador – representação interna.

Tipos de constantes:

- Números inteiros – números sem parte decimal.
- Números reais – números com parte decimal.
- Valores lógicos – True ou False.

- Cadeia de caracteres – sequência de caracteres delimitados por plicas. O comprimento da cadeia é o número de caracteres do seu conteúdo.

→ Uma **expressão composta**;

Uma expressão composta corresponde ao conceito de aplicação de uma operação a operandos, através de operações embutidas (pré-definidas/primitivas). Uma expressão composta é constituída por um operador e por um certo número de operandos. Os operadores podem ser unários (se apenas têm um operando, por exemplo, o operador not ou o operador – (simétrico)) ou binários (se têm dois operandos, por exemplo, + ou *).

Para evitar ambiguidades em relação à ordem de aplicação dos operadores numa expressão, o Python utiliza duas regras:

- Lista de prioridade de operadores;
- Ordem de aplicação dos operadores quando se encontram com a mesma prioridade – aplicam-se da esquerda para a direita.

Prioridade	Operador
Máxima	Aplicação de funções
	`not`, `-` (simétrico)
	`*`, `/`, `//`, `%`
	`+`, `-` (subtracção)
	`<`, `>`, `==`, `>=`, `<=`, `!=`
	`and`
Mínima	`or`

→ Um **nome**;

→ Uma **aplicação da função**.

2.2. Tipos elementares de informação

Um tipo de informação é caracterizado por um conjunto de entidades (valores) e um conjunto de operações aplicáveis a essas entidades. Ao conjunto de entidades dá-se nome de domínio do tipo. Cada uma das entidades do domínio do tipo é designada por elemento tipo.

Os tipos de informação podem dividir-se em dois grupos: tipos elementares e tipos estruturados.

→ Os **tipos elementares** são caracterizados pelo facto de as suas constantes (os elementos do tipo) serem tomadas como indecomponíveis (ao nível da utilização do tipo). [Ex.: "verdadeiro", "falso"].

→ Os **tipos estruturados** são caracterizados pelo facto de as suas constantes seres constituídas por um agregado de valores.

Como tipos elementares, existem, entre outros, o tipo inteiro, o tipo real e o tipo lógico.

→ Tipo inteiro – designado por ***int***, são números sem parte decimal, podendo ser positivos, negativos ou zero.

→ Tipo real – designados por ***float***, são números com parte decimal, podendo ser representados com notação decimal ou notação científica.

→ Tipo lógico – designado por ***bool***, apenas pode assumir dois valores Tru (verdadeiro) ou False (falso). As operações que se podem efetuar sobre os valores lógicos, produzindo valores lógicos, são de dois tipos, as operações unárias e as operações binárias.

- Operações unárias – produzem um valor lógico a partir de um valor lógico. Existe uma operação unária em Python, *not*. A operação *not* muda o valor lógico.
- Operações binárias – aceitam dois argumentos do tipo lógico e produzem um valor do tipo lógico. Entras as operações encontram-se as operações lógicas tradicionais correspondentes à conjunção e à disjunção. A conjunção *and* tem o valor True, apenas se ambos os seus argumentos tiverem valor True. A disjunção *or* tem valor False, apenas se ambos os seus argumentos têm valor False.

2.3. Nomes e atribuição

A instrução entre um nome e um valor é realizada através da instrução de atibuição. A instrução de atribuição em Python recorreà oeração embutida = (operador de atribuição).

Os nomes são utilizados para representar entidades usadas pelos programas.

⟨nome⟩ ::= ⟨nome simples⟩ |
⟨nome indexado⟩ |
⟨nome composto⟩

A instrução de atribuição apresenta duas formas distintas: atribuição simples (a que corresponde à primeira linha da expressão BNF que define <instrução de atribuição>) e atribuição múltipla (que corresponde à segunda linha da expressão BNF que define <instrução de atribuição>).

A instrução de atribuição não devolve valores, mas sim altera o valor de um nome. Ao associar um nome a um valor (ou nomear o valor) , o Python passa a "conhecer" esse nome, mantendo uma memória desse nome e do valor que lhe está associado. Esta memória correspondente à associação de nomes a valores (entidades computacionais) tem nome de ambiente. Um amibente contém asociações para todos os nomes que o Python conhece.

Ao executar uma instrução de atribuição, so o nome não existir no ambiente, o Pyhton insere o nome no ambiente, associando-o ao respetivo valor; se o nome já existir no ambiente, o Pyhton substitui o seu valor pelo valor da expressão. Deste comportamento, podemos concluir que, num ambiente, o mesmo nome não pode estar associado a dois valores diferentes.

Um ambiente é representado por um retangulo cinzento, dentro do qual aparecem associações de nomes a entidades computacionais. Cada associação contém um nome, apresentado no lado esquerdo, ligado por uma seta ao seu valor.

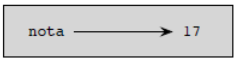


Figura 2.2: Representação de um ambiente.

Figura 2.3: Ambiente resultante da execução de `nota = nota + 1`.

A instrução de atribuição é a primeira instrução do Python que considerámos. Ao passo que uma expressão tem um valor e, consequentemente quando nos referimos às acções realizadas pelo Python para calcular o valor de uma expressão dizemos que a expressão é avaliada, uma instrução não tem qualquer valor mas causa a realização de certas acções, por exemplo a atribuição de um nome a uma variável. Por esta razão, quando nos referimos às acções efectuadas pelo Python associadas a uma instrução dizemos que a instrução é executada.~

2.4. Predicados e condições

Uma operação que produz resultados do tipo lógico chama-se predicado. Uma expressão cujo valor é do tipo lógico chama-se uma condição. As condições podem ser combinadas através de operações lógicas.

Nota: Se o valor da expressão for zero ou False, esta é considerada como falas, em caso contrário, é considerada como verdadeira.

2.5. Comunicação com o exterior

2.5.1. Leitura de dados

O Python fornece uma operação de leitura de dados, a função input. Esta função tem a sintaxe:

```
<leitura de dados>::= input ( )|
                      input (<informação>)
<informação>::= <cadeia de caracteres>
```

O valor da função input é a cadeia de caracteres cujo conteúdo é a sequencia de caracteres encontrada durante a leitura.

Um caráter de escape é um caráter mão gráfico com um significado especial para um meio de escrita.

Carácter escape	Significado
\\	Barra ao contrário (\)
\'	Plica (')
\"	Aspas (")
\a	Toque de campainha
\b	Retrocesso de um espaço
\f	Salto de página
\n	Salto de linha
\r	"Return"
\t	Tabulação horizontal
\v	Tabulação vertical

Tabela 2.8: Alguns caracteres de escape em Python.

A função eval recebe uma cadeia de caracteres e devolve o resultado a avaliar essa cadeira de caracteres como sendo uma expressão.

```
<função avaliação>::= eval(<cadeia de caracteres>)
```

2.5.2. Escrita de dados

As operações escritas de dados permitem transmitir a informação do programa para o exterior.

Em Python existe uma função embitida, print, com sintaxe definida pelas seguintes expressões em notação BNF:

```
<escrita de dados>::= print ( )|
                      print (<expressões>)
<expressões>::= <expressão>|
                <expressão>, <expressões>
```

2.6. Programas, instruções e sequenciação

Em notação BNF, um programa em Python, também é conhecido por um guião, é definido do seguinte modo:

<programa em Python>::= <definição>* <instruções>

Um programa não contém directamente expressões, aparecendo estas associadas a definições e a instruções.

As linguagens de programação fornecem estruturas que permitem especificar qual a ordem de execução das instruções do programa.

Ao nível da linguagem máquina, existem dois tipos de estruturas de controle: a sequenciação e o salto. A sequenciação especifica que as instruções de um programa são executadas pela ordem em que aparecem no programa. O salto especifica a transferência da execução para qualquer ponto do programa.

2.7. Seleção

A instrução if permite a selecção entre duas ou mais alternativas. Dependendo do valor de uma condição, esta instrução permite-nos seleccionar uma de umas ou mais instruções para serem executadas. A sintaxe da instrução if é definida pelas seguintes expressões em notação BNF:

⟨instrução if⟩ ::= `if` ⟨condição⟩: CR
⟨instrução composta⟩
⟨outras alternativas⟩*
{⟨alternativa final⟩}

⟨outras alternativas⟩ ::= `elif` ⟨condição⟩: CR
⟨instrução composta⟩

⟨alternativa final⟩ ::= `else:` CR
⟨instrução composta⟩

⟨instrução composta⟩ ::= TAB+ ⟨instruções⟩ TAB-

⟨condição⟩ ::= ⟨expressão⟩

Nota: CR é um símbolo terminal que corresponde ao símbolo obtido carregando na tecla "Return" do teclado, ou seja, CR corresponde ao fim de uma linha.

Ao encontrar uma instrução desta forma:

```
if ⟨cond1⟩:
   ⟨instruções1⟩
elif ⟨cond2⟩:
   ⟨instruções2⟩
elif ⟨cond3⟩:
   ⟨instruções3⟩
  ⋮
else:
   ⟨instruçõesf⟩
```

O Python começa por avaliar a expressão <$cond_1$>. Se o seu valor for True, as instruções correspondentes a <$instruções_1$> são executadas e a execução da instrução termina; se o seu valor for False, o Python avalia a expressão <$cond_2$>. Se o seu valor for treu, as instruções correspondentes a <$instruções_2$> são executadas e a execução if termina. E assim sucessivamente. Se todas as condições forem falsas, o Python executa as instruções correspondentes a <$instruções_f$>.

2.8. Repetição

Em programação, uma sequencia de instruções executadas repetitivamente é chama um ciclo. Um ciclo é constituído por uma sequência de instruções, o corpo do cluclo, e por uma estrutura que controla a execução dessas instruç~eos, especificando quantas vezes o ciclo é executado.

A instrução while permite especificar a execução repetitiva de um conjunto de instruções enquanto uma determinada expressão do tipo lógico tiver valor verdadeiro,

⟨instrução while⟩ ::= while ⟨condição⟩ : CR
⟨instrução composta⟩

No corpo do ciclo pode ser utilizada uma outra instrução existente em Python, a instrução break. A instrução break apenas pode aparecer dentro do corpo de um ciclo. Ao encontrar uma instrução break, o Python termina a execução do ciclo, independentemente do valor da condição que o controla.

```
<instrução break>::= break
```

Existem dois aspetos importantes a lembrar relativamente à instrução while:

→ De um modo geral, o número de vezes que o corpo do ciclo é executado não pode ser calcula antecipadamente: a condição que especifica o término do ciclo é testada durante a execução do próprio ciclo, sendo impossível saber de antemão o que vai prosseguir a avaliação.

→ Pode acontecer que o corpo do ciclo não seja executado nenhuma vez. Efeito, a semântica da instrução while especifica que o valor da expressão que controla a execução do ciclo é calculado antes do início da execução do ciclo. Se o valor inicial desta expressão é False, o corpo do ciclo não é executado.

Nota: a instrução for executa o ciclo, mesmo que o valor inicial da expressão seja False.

3. Funções

A possibilidade de agrupar informação e de a utilizar está ligada aos conceitos de função, procedimento e subprograma (dependendo da linguagem de programação). O Python utiliza a designação "função".

Note-se que a utilização de funções tem dois apetos distintos: a definição da função e a aplicação da função.

→ A definição da função é feita fornecendo um nome (ou designação para a função, uma indicação das suas variáveis (ou argumentos) e a indicação de um processo de cálculo para os valores da função, por exemplo, f(x)=x*x;

→ A aplicação da função é feita fornecendo o nome da função e um elemento do seu domínio para o qual se pretende calcular o seu valor, por exemplo, f(5).

3.1. Definição de funções em Python

Para definir funções em Python, é necessário indicar o nome daa função, os seus argumentos (designados por parâmetros formais) e o processo de cálculo (algoritmo) dos valores da função (designado por corpo da função).

⟨definição de função⟩ ::= def ⟨nome⟩ (⟨parâmetros formais⟩) : CR
TAB+ ⟨corpo⟩ TAB-

⟨parâmetros formais⟩ ::= ⟨nada⟩ | ⟨nomes⟩

⟨nomes⟩ ::= ⟨nome⟩ |
⟨nome⟩ , ⟨nomes⟩

⟨nada⟩ ::=

⟨corpo⟩ ::= ⟨definição de função⟩* ⟨instruções em função⟩

⟨instruções em função⟩ ::= ⟨instrução em função⟩ CR |
⟨instrução em função⟩ CR ⟨instruções em função⟩

⟨instrução em função⟩ ::= ⟨instrução⟩ |
⟨instrução return⟩

⟨instrução return⟩ ::= return |
return ⟨expressão⟩

Ex.:

```
def quadrado(x):
    return x*x
```

→ antes de terminar retorna o valo x

3.2. Aplicação de funções em Python

Uma vez definida, uma função pode ser usada do mesmo modo que as funções embutidas, fornecendo ao Python, numa expressão o nome da expressão, o nome da função seguido do número apropriado de argumentos (os parâmetros concretos).

Para calcular o valor de uma função, o Python utiliza a seguinte regra:

- Avalia os parâmetros concretos (por qualquer ordem);
- Associa os parâmetros formais da função com os valores dos parâmetros concretos calculados no passo anterior;
- Cria um novo ambiente, um ambiente local à função, definido pela associação entre os parâmetros formais e os parâmetros concretos. No ambiente local executa as instruções correspondentes ao corpo da função. O ambiente local apenas existe enquanto a função estiver a ser executada e é apagado pelo Python quando termina a execução da função.

A associação de nomes a entidades computacionais tem o nome de frame / quadro (quadrado → função).

Podemos dizer que quando uma função é chamada, é criado um ambiente local, o qual corresponde a uma associação entre os parâmetros formais e os concretos. Este ambiente local desaparece no momento em que termina a avaliação da função que deu origem à sua criação.

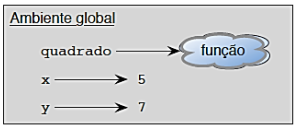


Figura 3.1: Ambiente global com os nomes `quadrado`, `x` e `y`.

Nota:

- O quadro global não tem nenhum quadro envolvente.
- Como estão em ambientes diferentes, não importa que a variável tenha a mesma terminação, porque são variáveis diferentes.

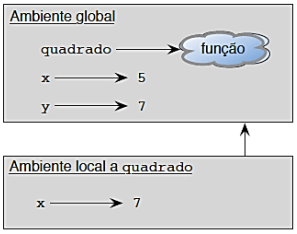


Figura 3.2: Ambiente criado durante a avaliação de `quadrado(y)`.

3.3. Abstração procedimental

A abstracção procedimental consiste em dar um nome à sequência de acções que serve para atingir um objetivo (e, consequentemente abstrair do modo como as funções realizam as suas tarefas), e em utilizar esse nome, sempre que desejarmos atingir esse objetivo sem termos de considerar explicitamente cada uma das acções

individuais que as constituem. Assim, a abstracção procedimental são pedaços de código, que realizam uma tarefa e que recorda algo.

Consiste em abstrair de como as funções realizam as suas tarefas, concentram-se apenas na tarefa que as funções realizam. Ou seja, a separação do "como" de "que".

Figura 3.3: O conceito de caixa preta.