APOSTILA PYTHON

Física Computacional I

Professor Felipe Fanchini

Baseado na Apostila Fortran Francisco Lavarda [1]

Conversão Fortran-Python por Felipe Mahlow [2]

Capítulo 0: Notas Introdutórias

[D] Convenções usadas nestas notas:

- [D] Definição.
- [C] Comentário.
- [CP] Comando Python.
- [SO] Comando do Sistema Operacional usado.
- [ET] Comando de edição do editor de texto usado.
- [E] Exemplo.
- [P] Problema.
- [A] Atividade.
- [G] Gíria da área (jargão).
- ＜tecla＞ significa uma dada tecla que deve ser pressionada.

[C] "comandos que você deve digitar" estarão sempre entre aspas.

Capítulo 1: Noções Preliminares

1.1. Introdução

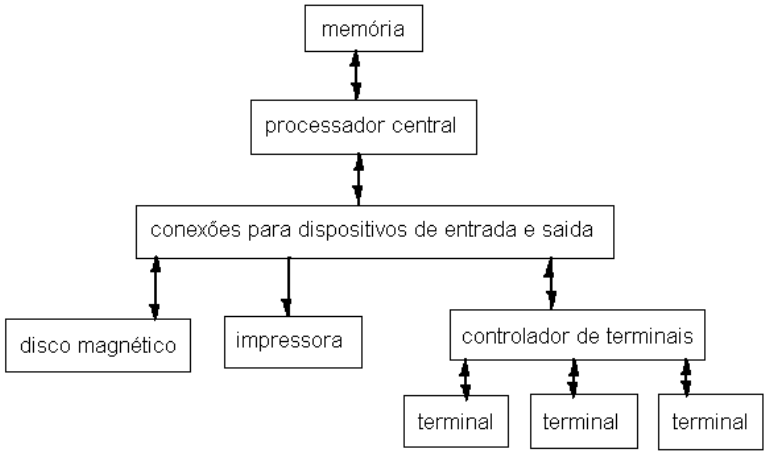
[C] "O que todo computador pode realmente fazer é seguir ordens muito simples, as quais foram cuidadosamente consideradas e refletidas por um programador e escritas em uma linguagem de programação, como Python."

1.2. Computadores

[C] Há somente 4 tipos de instruções que um computador pode realizar:

1. Armazenamento e recuperação de informações da memória;
2. Cálculos;
3. Entrada e saída de dados;
4. Controle de programa.

[C] Esquema de um computador:



1.3. Algoritmos

[D] Um "algoritmo" é um conjunto de instruções passo-a-passo para resolver um problema.

[D] Um algoritmo correto deve possuir 3 qualidades:

1. Cada passo no algoritmo deve ser uma instrução que possa ser realizada.
2. A ordem dos passos deve ser precisamente determinada.
3. O algoritmo deve ter fim.

[A] Elaboração em grupos de 3 de um algoritmo para que um dos elementos do grupo desempenhe a seguinte tarefa:

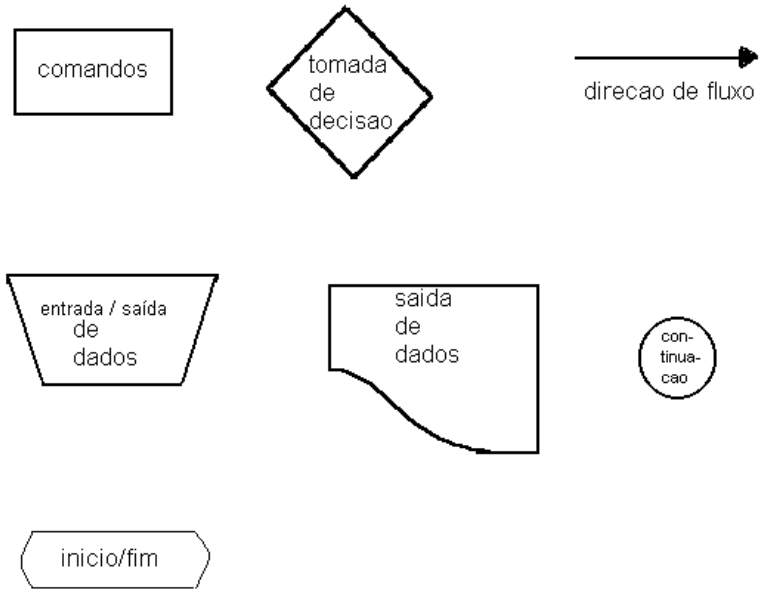
- Sair de uma posição na sala de aula, que será determinada pelo professor, até o interruptor das luzes e providenciar o desligamento destas.

As operações corporais possíveis devem ser determinadas antes do começo do trabalho em grupo, em conjunto entre os alunos e o professor. Após o término do trabalho dos grupos, o professor escolherá aleatoriamente trabalhos, os lerá em voz alta, determinando se os passos do algoritmo estão contidos no conjunto de operações permitidas. Se isto for verdade, um dos elementos do grupo lerá em voz alta os passos que serão executados por outro membro do grupo. Sugestões de operações permitidas: caminhar, girar o corpo (graus e sentido), levantamento de braço (graus e sentido), movimento de mão, cegueira do executor, etc.

1.4. Diagramas de Fluxo

[D] "Diagramas de Fluxo" fornecem uma representação gráfica de um algoritmo.

[D] Principais símbolos de diagramas de fluxo:



1.5. Linguagens de Programação

[D] Um "programa de computador" é um algoritmo escrito numa linguagem de programação, como o Python.

[C] Existem 3 níveis de linguagens de programação:

1. Linguagens de baixo nível (ou de máquina).
2. Linguagens de nível intermediário (ou simbólica montadora).
3. Linguagens de alto nível (ou de compiladores).

1.5.1 Linguagem de Máquina

[D] Cada instrução é constituída de 2 partes:

código de operação operando(s)

[D] O "código de operação" instrui o computador para realizar uma função específica: adição, subtração, leitura, escrita, comparação, movimento de dados, etc...

[D] O(s) operando(s) especifica(m) o endereço do dado a ser operado, endereço do dispositivo de entrada/saída, etc...

[E] Exemplo do trecho principal de um programa em linguagem de máquina para calcular R = ((5A+16B)/C) - D, onde os valores de A, B, C, D e R estão armazenados nos endereços 1000, 1004, 1012, 1020 e 2050 da memória.

Código da Operação	Operando	Comentário
14	1000	carrega A no acumulador
12	=5	multiplica o acumulador por 5
15	3000	armazena conteúdo acumulado no endereço 3000
14	1004	carrega B no acumulador
12	=16	multiplica acumulador por 16
10	3000	adiciona acumulador com conteúdo de 3000
13	1012	divide acumulador pelo conteúdo de 1012
11	1020	subtrai do acumulador o conteúdo de 1020
15	2050	armazena acumulador em 2050

[C] Na realidade, um programa real em linguagem de máquina, para ser compreendido por ela, geralmente é escrito no sistema binário.

1.5.2. Linguagem Simbólica Montadora

[C] As linguagens simbólicas procuram facilitar a tarefa do programador, criando mnemônicos para as operações.

[C] Os programas montadores traduzem um programa fonte (ou seja, em linguagem simbólica) em um programa objeto (ou seja, em linguagem de máquina).

[E] O trecho de programa visto acima, ficaria em uma linguagem simbólica parecido com:

Operação	Operando	Comentário
CAR	A	carrega A
MUL	5	multiplica por 5
ARM	TMP	armazena o resultado em TMP
CAR	B	carrega B
MUL	16	multiplica por 16
ADI	TMP	adiciona o resultado com o conteúdo de TMP
DIV	C	divide o resultado por C
SUB	D	subtrai o resultado de D
ARM	R	armazena o resultado em R

1.5.3. Linguagens de Compiladores

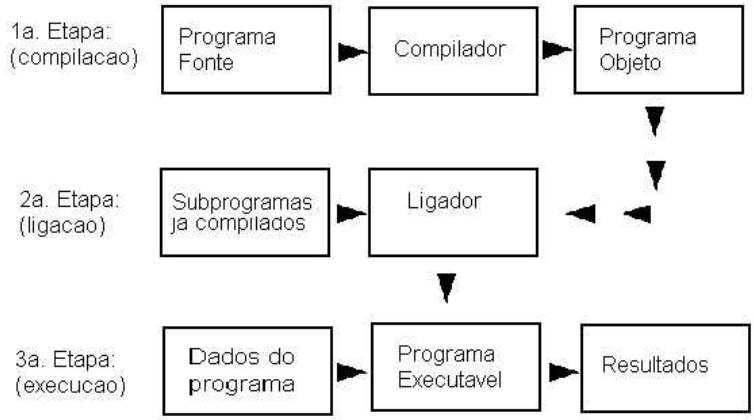
[C] Uma linguagem de compilador (ou de alto nível) tem por objetivo permitir ao programador se utilizar de uma linguagem muito próxima àquela utilizada no ambiente no qual se coloca a tarefa a ser realizada.

[C] Os programas compiladores traduzem um programa fonte, escritos em uma linguagem de alto nível, em um programa objeto em linguagem de máquina.

[E] O trecho de programa visto acima, ficaria na linguagem de alto nível Python idêntico a:

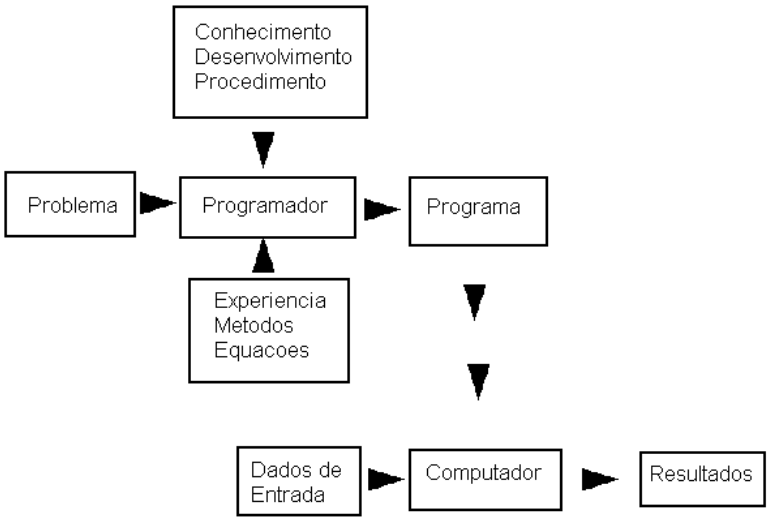
$$R = ((5.0 * A + 16.0 * B)/C) - D$$

[C] Etapas do processamento de um programa em linguagem de alto nível:



*Python especificamente se trata de uma linguagem interpretada, então existem pequenas diferenças com relação ao fluxograma acima.

1.6.Passos no Desenvolvimento de Programas



Capítulo 2: A Linguagem Python: Conceitos Básicos

2.1. Introdução

[C] O nome Python teve a sua origem no grupo humorístico britânico Monty Python, criador do programa Monty Python's Flying Circus, embora muitas pessoas façam associação com o réptil do mesmo nome.

[C] É uma linguagem de programação de alto nível, interpretada de script, imperativa, orientada a objetos, funcional, de tipagem dinâmica e forte.

[C] Foi lançada por Guido van Rossum em 1991. Atualmente, possui um modelo de desenvolvimento comunitário, aberto e gerenciado pela organização sem fins lucrativos Python Software Foundation.

2.2. Definições da Linguagem

D] Programa Principal é uma unidade que pode chamar outras unidades de programa, mas que não pode ser chamado por estas. O programa principal é quem recebe o controle da execução no início da fase de execução.

[D] Subprograma é uma unidade de programa que é chamada por outra unidade de programa. Usualmente um subprograma recebe parâmetros de entrada e retorna parâmetros de saída. No caso de Python, utilizamos funções.

[D] Função Intrínseca é uma função suprida pelo interpretador e que realiza operações sobre números ou caracteres.

[D] Programa Executável é um programa já em linguagem de máquina que pode ser executado pelo computador. Consiste de um programa principal e subprogramas.

[D] Comando Executável é aquele que efetua cálculos, testes, providencia entrada ou saída de dados, altera o fluxo de execução ou atribui valores a variáveis.

[D] Comando Não-Executável é aquele que descreve as características de uma unidade de programa, dos dados ou de informações de edição.

[D] Arquivos de Dados são unidades de entrada e saída de dados, passíveis de serem lidos/escritos por um programa.

2.3. Itens sintáticos do Python

[D] Sintaxe é a parte da gramática que estuda a disposição das palavras na frase e a das frases no discurso.

[D] Constantes são valores ou números que ocorrem num programa.

[D] Nome simbólico é uma sequência de uma ou mais letras ou dígitos, o primeiro dos quais deve ser uma letra.

[D] Variáveis são valores ou números ou conjuntos de caracteres que ocorrem num programa e que podem sofrer variações durante a execução do programa. Na realidade uma variável é o nome de uma localização da memória.

[D] Palavra chave é uma sequência de letras que identificam exclusivamente comandos Python.

[D] Operador é um ente similar a um operador matemático.

[C] Valores são obtidos da memória quando eles são usados no lado direito do sinal de igual.

[C] Observe que o sinal de "=" não indica uma igualdade e sim uma atribuição.

[C] O Python permite que você defina diferentes tipos de variáveis. Usa-se variáveis inteiras para armazenar números inteiros, variáveis reais para números reais, variáveis complexas para números complexos, variáveis caracter para palavras e frases, etc.

[C] Você pode definir os tipos de variáveis no seu programa através de "comandos de definição de tipo".

[CF] Existem vários comandos Python para a definição de tipo de variável, entre os quais "str", "int" e "float".

[C] Os cálculos podem ser feitos através de comandos de atribuição, escrevendo-se uma "expressão" no lado direito do sinal de igual.

[C] Símbolos de operações matemáticas (também conhecidos como "operadores aritméticos"): adição, "+"; subtração, "-"; multiplicação, "*"; divisão, "/"; potenciação, "**".

[C] Pode-se usar parênteses para agrupar termos em expressões Python, como é feito em álgebra.

2.4. Caracteres usados no Python

[D] O conjunto de caracteres Python é constituído de: letras (ou caracteres alfabéticos), dígitos (ou caracteres numéricos) e símbolos especiais como =, +, -, /, *, etc.

2.5. Uma visão geral do Python

[E] Exemplo de um programa para calcular a média das notas de uma prova de uma turma e determinar o número de notas acima e abaixo de 5,0. Vamos usar a equação abaixo para calcular a média da turma:

$$MT = \frac{\sum_{i=1}^{NT} nota(i)}{NT}$$

sendo que: MT = média da turma, nota(i) = nota do aluno i, NT = número total de alunos.

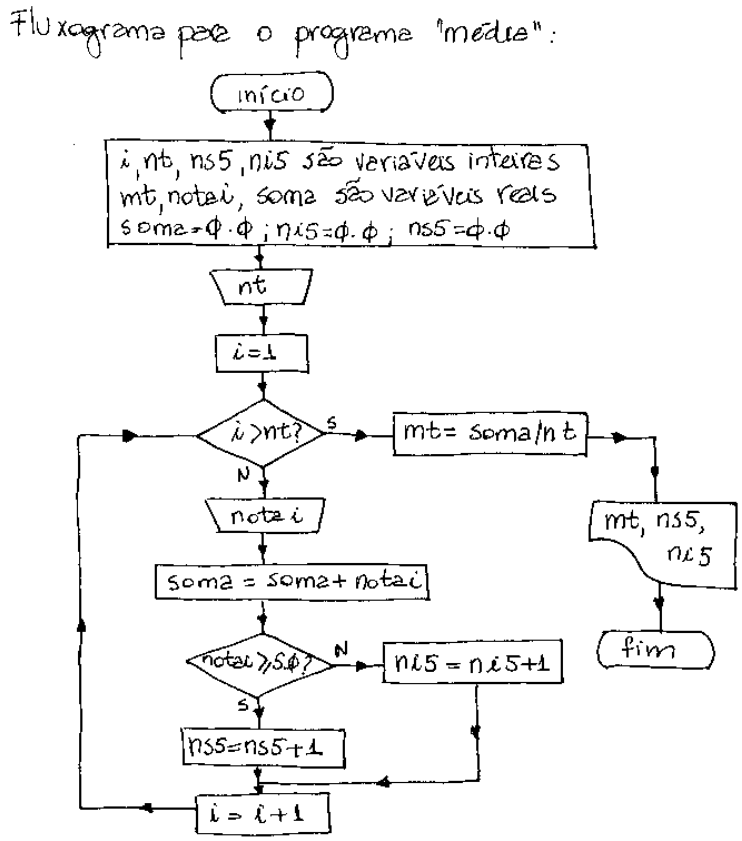
O código do programa poderia ser:

In [19]:

```
nt = int(input("Digite o número total de alunos: "))
ns5 = ni5 = soma = 0
for i in range(0, nt):
    nota = float(input("Digite a nota do aluno " + str(i+1) + ": "))
    soma += nota
    if nota >= 5:
        ns5 += 1
    else:
        ni5 += 1
print("A média da turma foi: ", (soma/nt))
print("Notas superiores ou iguais a 5.0: ", ns5)
print("Notas inferiores a 5.0: ", ni5)
```

```
Digite o número total de alunos: 3
Digite a nota do aluno 1: 10
Digite a nota do aluno 2: 8
Digite a nota do aluno 3: 3
A média da turma foi:  7.0
Notas superiores ou iguais a 5.0:  2
Notas inferiores a 5.0:  1
```

[C] Veja figura a seguir contendo o fluxograma para o programa media.py.



[C] Uma "variável" Python é um nome para uma localização de memória. Variáveis no exemplo acima: nt, ns5, ni5, mt, nota, soma, i.

[C] O "valor" de uma variável é a informação armazenada na localização de memória que é representada pela variável.

[D] Um "comando de atribuição" é o tipo de comando que permite armazenar um valor em uma variável. Comandos de atribuição no exemplo acima:

In [20]:

```
ns5 = 0
ni5 = 0
soma = 0
```

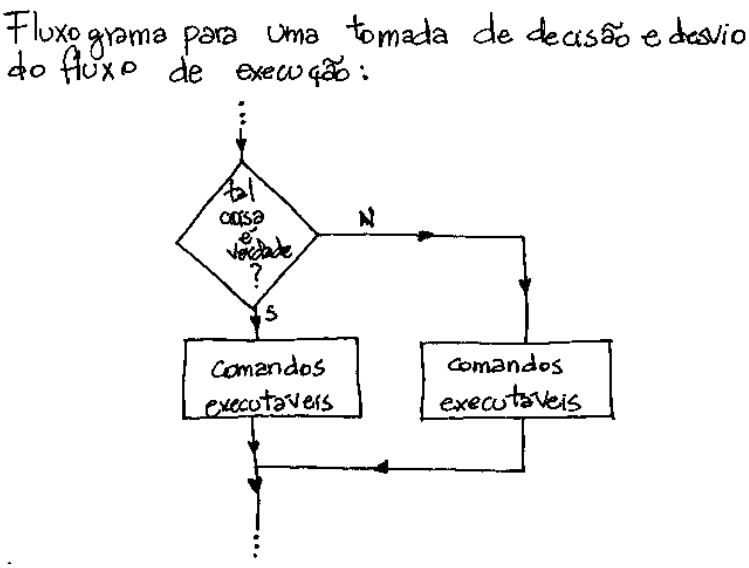
[C] Quando você atribui um valor a uma variável, o valor anteriormente armazenado é destruído.

[C] Os comandos num programa Python são, normalmente, executados na ordem em que são escritos. Entretanto, pode-se alterar esta ordem de execução através dos "comandos de controle de fluxo".

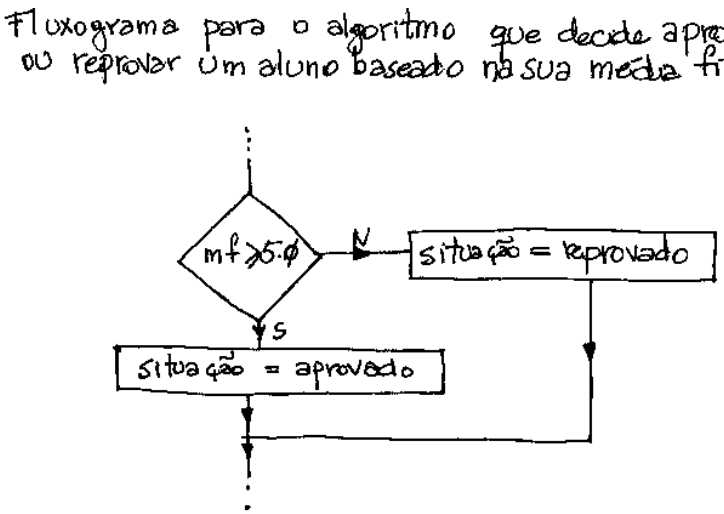
[C] O espaçamento (Identação) identifica o que deve ser executado dentro dos comandos de controle de fluxo.

[CF] Um comando de controle que representa uma decisão é a construção "if/ elif/ else".

[C] Veja figura a seguir para uma tomada de decisão e desvio do fluxo de execução.



[E] Veja figura a seguir para o fluxograma para o algoritmo que decide aprovar ou reprovar um aluno, baseado na sua média final.



[C] Comandos Python para o fluxograma anterior:

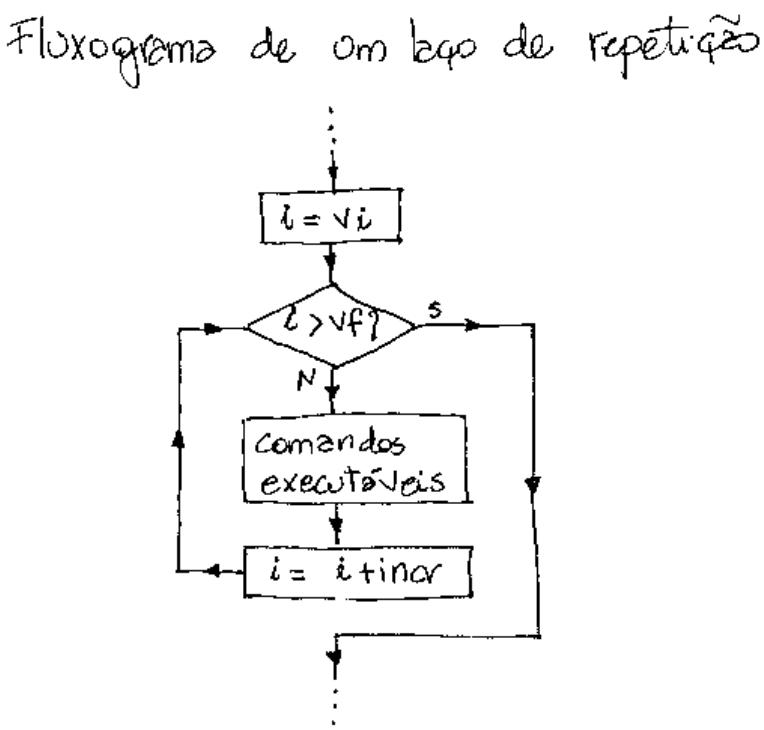
In [4]:

```
mf = int(input("Digite a média final do aluno: "))
if mf >= 5:
    situacao = "Aprovado"
else:
    situacao = "Reprovado"
print(situacao)
```

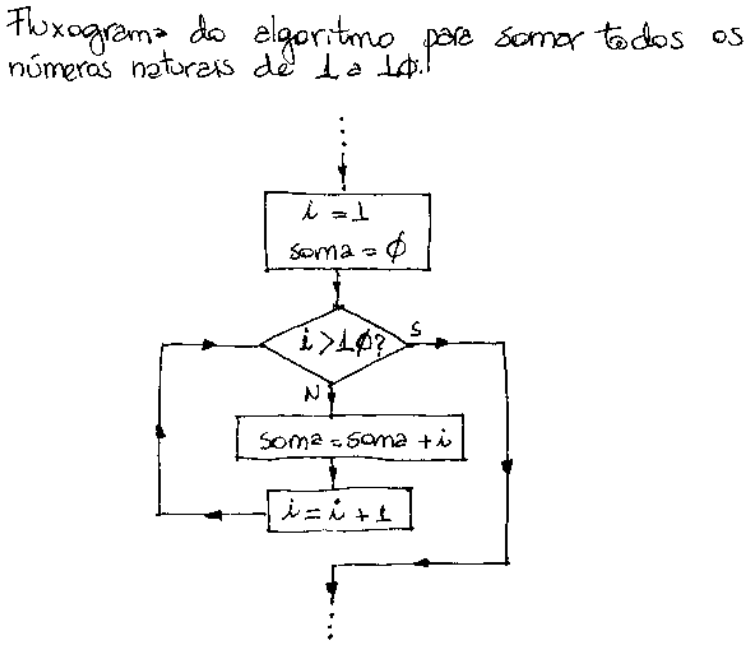
```
Digite a média final do aluno: 5
Aprovado
```

[CF] Um comando de controle que representa um laço é o comando "for".

[C] Veja a seguir um fluxograma de um laço de repetição.



[E] Veja a seguir o fluxograma para o algoritmo para somar todos os números naturais de 1 a 10.



[C] Comandos Python para o fluxograma anterior:

[C] "+=" significa acrescentar o valor "à direita da equação" ao valor já contido na variável. É similar à escrita soma = soma + i no caso abaixo.

[C] Atenção! Qualquer intervalo delimitado em Python se inicia com no valor à esquerda e finaliza no antecessor do valor à direita. Por este motivo, quando declaramos o range (alcance) abaixo, ele é dado como (1,11), desta forma estamos tendo como alvo os valores de 1 a 10.

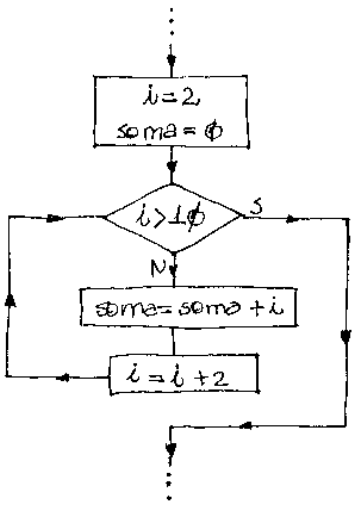
In [27]:

```
soma = 0
for i in range(1,11):
    soma += i
print(soma)
```

```
55
```

[E] Veja a seguir o fluxograma para o algoritmo para somar todos os números naturais pares de 1 a 10.

Fluxograma do algoritmo para somar todos os números naturais pares de 1 a 10:



[C] Comandos Python para o fluxograma anterior:

In [30]:

```
soma = 0
for i in range(0, 11, 2):
    soma += i
print(soma)
```

```
30
```

Capítulo 3: Constantes, Variáveis e Conjuntos

3.1 Introdução

[C] Este capítulo trata de definir corretamente os componentes básicos dos comandos em Python. Alguns já vimos no capítulo anterior, como "itens de sintaxe".

[D] "Constantes" são valores fixos, tais como números. Estes valores não podem ser alterados pelos comandos do programa. Exemplos: 3.0, 3, 'palavra'.

[C] "Variáveis" são nomes simbólicos que representam valores armazenados na memória do computador. Estes valores podem ser alterados pelos comandos do programa.

[D] "Conjuntos" são grupos de variáveis, cujos valores são armazenados adjacentemente e podem ser referenciados individualmente através de um nome simbólico com um índice. São conhecidos também como "variáveis indexadas" (VI).

[D] Um "operador" é um símbolo específico para uma determinada operação. Exemplo: + (soma).

[D] "Expressões" são combinações de constantes, variáveis, elementos de conjuntos e operadores. Exemplo: 3.0*var1+var2.

[E] No "comando de atribuição" abaixo:

$$fat = 3 * 4 * (B + 2.5)$$

temos a "expressão" $3 * 4 * (B + 2.5)$. Nesta expressão temos três "constantes" (3, 4 e 2.5), uma "variável" (B), dois "operadores" (* e +) e parênteses. Vemos também que temos duas constante inteiras (3 e 4) e uma real (2.5); a variável B depende de como tenha sido definida.

[E] No trecho de programa abaixo:

In [11]:

```
v0y = float(input("Digite a velocidade inicial: "))
g = 9.8
dt = 0.1
t = list()
y = list()
for i in range(0, 5):
    t.append(i*dt)
    y.append(v0y*t[i] - 0.5*g*t[i]**2)
print(y, t)
```

```
Digite a velocidade inicial: 5
[0.0, 0.45099999999999996, 0.8039999999999999, 1.0590000000000002, 1.2159999
999999997] [0.0, 0.1, 0.2, 0.30000000000000004, 0.4]
```

vemos o uso das "variáveis indexadas" t(i) e y(i).

3.2. Constantes

[D] Uma "constante" é uma quantidade fixa e invariável.

[D] O Python distingue três classes de constantes: numéricas, lógicas e cadeias de caracteres.

[D] As constantes numéricas que mais nos interessam são:

- Inteiras: para números inteiros decimais (escritos sem o ponto decimal).
- Reais: para números decimais (ou fracionários).
- Complexos: para números complexos.

[D] As constantes booleanas (lógicas) podem ser:

- True : representa o valor "verdade".
- False : representa o valor "falso".

[D] As constantes Strings (cadeias de caracteres) são uma seqüência de caracteres alfanuméricos e/ou especiais sempre entre aspas.

[C] Podemos ainda, armazenar conjuntos de constantes ou mesmo variáveis, utilizando listas, tuplas ou dicionários.

3.3. Variáveis

[D] Uma variável possui um nome e um tipo, podendo assumir diversos valores.

[D] Regras para nomes de variáveis:

1. Os nomes devem começar com uma letra ou caractere especial.
2. Os nomes podem conter letras e dígitos.

[D] Tipos de variáveis: inteiras, reais, complexas, lógicas e strings, que possuem características similares às constantes.

3.4. Conjuntos

[C] Nesta seção nos ocuparemos dos conjuntos de variáveis, ou variáveis indexadas (VI), já definidos no início deste capítulo.

[C] Uma VI possui um nome, um tipo e um conjunto de índices (ou um único índice).

[D] "Vetor" é uma VI que possui um único índice, e por isto é dito uma VI unidimensional.

[E] A variável t no trecho de programa abaixo é um vetor:

In [15]:

```
t[0] = 0.1
t[1] = 0.2
t[2] = 0.3
print(t)
```

```
[0.1, 0.2, 0.3, 0.30000000000000004, 0.4]
```

[D] "Matriz" é uma VI que possui dois conjuntos de índices e por isto é dita bidimensional.

[E] A variável a no trecho de programa abaixo é uma "lista de listas", que representa uma matriz:

In [19]:

```
a = [[0, 1, 2], [1, 2, 3], [2, 3, 4]]
for i in range(0, 3):
    for j in range(0, 3):
        print(a[i][j])
```

```
0
1
2
1
2
3
2
3
4
```

- Representação Matricial da variável a:

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix}$$

- Forma de armazenamento na memória:

... a_{11} a_{21} a_{31} a_{12} a_{22} a_{32} a_{13} a_{23} a_{33} ...

[D] Dizemos que uma VI é uma matriz n-dimensional quando esta possui um conjunto de n índices.

Capítulo 4: Expressões

4.1. Introdução

[D] Uma "expressão" em Python é definida como uma combinação de itens sintáticos, isto é: uma expressão pode consistir de uma única constante, de uma única variável, de um único elemento de conjunto, ou uma combinação de constantes, variáveis, elementos de conjuntos, unidos com um ou mais operadores e, possivelmente, agrupados com parênteses.

[D] "Operadores" especificam as computações a serem realizadas sobre os valores dos elementos básicos.

[D] As expressões, no Python, são classificadas em aritméticas, caracteres, relacionais e lógicas.

4.2. Expressões Aritméticas

[D] Uma "expressão aritmética" é usada para expressar uma computação numérica. O cálculo desta expressão produz um valor numérico cujo tipo é inteiro, real, dupla precisão ou complexo.

[D] Os "operadores aritméticos" são definidos pela tabela abaixo:

Operador	Definição	Uso	Significado
**	potenciação	a**b	a elevado à potência b
*	multiplicação	a*b	a multiplicado por b
/	divisão	a/b	a dividido por b
+	adição	a+b	a mais b
-	subtração	a-b	a menos b
-	menos unário	-a	a com sinal invertido

[C] Em expressões contendo um único operando, o tipo do operando define o tipo de expressão.

[C] Em expressões com mais de um tipo de operando, o tipo da expressão depende dos tipos dos operandos e dos operadores.

[D] Para todas as operações (com exceção da potenciação):

1. Se os operandos forem do mesmo tipo, este será o tipo da expressão.
2. Se os operandos forem de tipos diferentes, o tipo da expressão será o tipo do operando de maior precedência, dada pela tabela abaixo:

Tipo - Precedência

1. inteiro
2. real
3. dupla precisão

4. complexo

[D] Regras para ordem de computação:

1. Parênteses podem ser usados para agrupar operandos em uma expressão. Qualquer expressão entre parênteses é calculada antes do cálculo da expressão da qual ela é uma parte. Parênteses mais internos são calculados antes.
2. Com exceção do uso dos parênteses, as operações são realizadas na seguinte ordem de precedência:

Operação	Precedência
+ e -	mais baixa
* e /	intermediária
**	mais alta

3. Para operadores de mesma precedência, o cálculo se desenvolve da esquerda para a direita.

4.3. Expressões Relacionais

[D] Uma "expressão relacional" é usada para comparar os valores de duas expressões aritméticas (ou os valores de duas expressões caractere).

[D] A comparação entre duas expressões aritméticas é feita através dos "operadores relacionais".

[C] O cálculo de uma expressão relacional produz um resultado lógico, com um valor "verdadeiro" ou "falso". A expressão é interpretada como "verdade" se os valores dos operandos satisfazem a relação matemática especificada pelo operador, e "falso" se eles não satisfazem a relação.

[D] "Operadores Relacionais" são aqueles definidos na tabela abaixo:

Operador	Definição	Uso	Significado
==	igual a	A==B	A é igual a B ?
!=	diferente de	A/=B	A é diferente de B?
<	menor que	A<B	A é menor a B?
<=	menor ou igual	A<=B	A é menor ou igual a B?
>	maior que	A>B	A é maior que B ?
>=	maior ou igual	A>=B	A é maior ou igual a B?

4.4. Expressões Lógicas

[D] Uma expressão lógica é usada para realizar um cálculo lógico.

[D] Os "operadores lógicos" são (listamos somente os principais):

Operador	Definição	Uso	Significado
not	negação	not A	se A é verdade, not A é falso

and	conjunção	A and B	para a expressão ser verdade, A e B precisam ser verdade
or	disjunção	A or B	para a expressão ser verdade, A ,ou B, precisa ser verdade

In [9]:

```
x1 = 1
x2 = 1
x = 1
if x1 == x and x2 == x:
    print("todos são iguais!")
```

```
todos são iguais!
```

In [10]:

```
escrever = True
if escrever:
    print("Olá mundo!")
```

```
Olá mundo!
```

[D] Regras para cálculos de expressões lógicas.

Parênteses podem ser usados e são calculados antes. Os operadores aritméticos são calculados em segundo lugar. Os operadores relacionais são calculados em terceiro lugar. As operações lógicas são realizadas na seguinte ordem de precedência:

Operador - Precedência

- not - mais alta
- and - intermediária
- or - mais baixa

Capítulo 5: Comandos de Atribuição

5.1. Introdução

[D] O "comando de atribuição" é usado para atribuir um valor a (ou definir) uma variável ou a um elemento de conjunto.

[C] Os comandos de atribuição calculam uma expressão e atribuem o valor resultante a uma variável ou elemento de conjunto.

[D] A forma de um comando de atribuição é:

```
variável=expressão
```

[D] Os comandos de atribuição são classificados como:

- aritmético;
- caractere;
- lógico.

[E] Exemplos:

- aritmético: $varnum = 3.0 * a/b$
- caractere: $varcar =" \ palavra \ "$
- lógico: varlog=True ou $varlog = a < b \ and \ b == c$

[C] Após o cálculo da expressão, o valor é convertido, se necessário, para o tipo de variável de acordo com a seguinte regra:

Tipo da variável	Valor atribuído
inteiro	a parte inteira da expressão
real	o valor real da expressão
complexo	o valor complexo da expressão

Capítulo 6: Comandos de Especificação: Tipos de Variáveis

6.1. Introdução

[D] Os comandos de declaração de tipo são: int(), float(), complex(), bool() e str().

6.2. Comandos de Especificação Explícita

Podem ser utilizados para especificar ou converter tipos de variáveis, desde que logicamente possível.

6.2.1. Comando INTEGER

[D] O comando int() é usado para especificar variáveis inteiras.

[CF] A forma geral do comando INTEGER é:

```
int(valor)
```

[E] Exemplos:

In [3]:

```
int("5")
```

Out[3]:

```
5
```

In [4]:

```
int(5.2)
```

Out[4]:

```
5
```

6.2.2. Comando FLOAT

[D] O comando float() é usado para especificar variáveis reais.

[CF] A forma geral do comando REAL é:

```
float(valor)
```

[E] Exemplos:

In [6]:

```
float("5")
```

Out[6]:

```
5.0
```

In [7]:

```
float(5.3)
```

Out[7]:

```
5.3
```

6.2.3. Comando COMPLEX

[D] O comando COMPLEX é usado para especificar variáveis do tipo complexo.

[C] A letra "j" é utilizada para definir a parte imaginária em Python.

[CF] A forma geral do comando COMPLEX é:

```
complex(valor)
```

[E] Exemplos:

In [8]:

```
complex(5)
```

Out[8]:

```
(5+0j)
```

In [10]:

```
complex(5 + 5j)
```

Out[10]:

```
(5+5j)
```

6.2.4. Comando BOOLEAN

[D] O comando BOOLEAN é usado para especificar variáveis do tipo lógico.

[CF] A forma geral do comando BOOLEAN é:

```
bool(valor)
```

6.2.5. Comando STRING

[D] O comando STRING é usado para especificar variáveis do tipo caractere.

[CF] A forma geral do comando STRING é:

```
str("nome")
```

[E] Exemplos:

In [15]:

```
str("Nome")
```

Out[15]:

```
'Nome'
```

In [16]:

```
str(5)
```

Out[16]:

```
'5'
```

Capítulo 7: Comandos de Controle de Fluxo e Programação Estruturada

7.1. Introdução

[C] Normalmente, a execução dos comandos num programa é feita na ordem em que os comandos são escritos: de cima para baixo, linha por linha, de acordo com a estrutura sequencial. Entretanto você pode usar comandos para transferir o controle de fluxo para outra parte da mesma unidade de programa ou para outra unidade de programa.

[C] O controle pode ser transferido somente para um comando executável.

7.2. Estruturas de Controle

[C] As estruturas básicas de controle são:

1. Estrutura sequencial.
2. Estrutura de tomada de decisão (if-elif-else).
3. Estrutura de laço:
4. Estrutura de laço repetição (for-loop).
5. faça-enquanto (while-loop).

7.3. Comandos IF

[CF] Os comandos IF transferem o controle do fluxo ou executam outro comando (ou um bloco de comandos) dependendo do resultado verdadeiro ou falso de uma expressão lógica contida no particular comando IF. Os três tipos de comandos IF são:

1. IF aritmético.
2. IF lógico.
3. IF bloco.

[CF] Os comandos elif e else, são também apresentados nesta seção, pois eles são usados somente em conjunto com um comando if bloco.

7.3.1. Comando IF lógico

[CF] O comando IF lógico calcula uma expressão lógica/relacional e executa ou ignora um comando executável contido no próprio IF, dependendo do valor (falso ou verdadeiro) dessa expressão.

[CF] A forma geral do comando IF lógico é:

```
if e:
    c
```

sendo que:

- "e" é uma expressão lógica ou relacional.
- "c" é o comando que será executado caso a expressão seja verdade.

[E] Vejamos como fica o fluxo do programa quando este chega a estas quatro linhas:

```
x=a-b
if a == b:
    x=a+b
y=x**2
```

Observe que nas últimas 3 linhas:

1. if(a.eq.b)
2. x=a+b
3. y=x**2

Então:

1. Se de fato a=b, teremos a seguinte ordem de execução: 1 >> 2 >> 3.
2. Se, por outro lado, a é diferente de b, teremos a ordem de execução: 1 >> 3.

7.3.2. Comandos IF bloco

[CF] O comando IF bloco permite a execução de um determinado bloco de comandos, dependendo do valor da(s) expressões lógica(s)/relacional(is) nele contido.

[CF] A forma geral do comando IF bloco é:

```
if e1:
   bloco1
elif e2:
   bloco2
elif e3:
   bloco3
...
else:
   bloco n
```

sendo que e1, e2,e3, ... são expressões lógicas/relacionais. Deve ficar claro que somente um dos blocos de comandos será executado.

[C] O menor IF bloco é a estrutura:

```
if e:
    bloco
```

Vemos que os comandos elif e else são opcionais.

[CF] O comando if permite a execução condicional de um bloco de comandos executáveis. É obrigatório usar em conjunção com o espaçamento (identação) para identificar onde o bloco acaba.

[CF] A forma geral do comando IF é:

```
if e:
```

sendo que "e" é uma expressão lógica/relacional (portanto produzindo um resultado verdadeiro ou falso).

[C] Se a expressão "e" for verdadeira, a execução passa para o próximo comando executável até que se encerre a identação. Se for falsa, o controle é transferido para o próximo comando.

[CF] O comando ELSE fornece uma rota alternativa para um comando IF ou ELIF. Sua forma geral é simplesmente:

```
else:
```

[CF] O comando ELSE é utilizado quando a condição do if acima não é satisfeita.

[CF] O comando ELIF combina as funções dos comandos ELSE e IF, pois fornece uma rota alternativa e torna possível uma estrutura IF bloco com mais de uma alternativa.

[CF] A forma geral do comando ELIF é:

```
elif e:
```

sendo que "e" é uma expressão lógica/relacional.

[C] Se a expressão for verdadeira, o controle passa para o próximo comando executável até o final da identação. Se for falsa, passa para o próximo comando.

[E] Exemplo:

In [3]:

```
nota = float(input("Digite a nota do aluno: "))
if nota >= 5:
    print("O aluno foi aprovado")
elif nota >= 3.5:
    print("O aluno foi reprovado, mas poderá fazer o exame")
else:
    print("o aluno foi reprovado e não poderá fazer o exame")
```

```
Digite a nota do aluno: 4
O aluno foi reprovado, mas poderá fazer o exame
```

7.3.3. Estruturas de IF bloco encaixados

In [4]:

```
nota = float(input("Digite a nota do aluno: "))
if nota >= 5:
    if nota >= 9.5:
        print("O aluno foi aprovado com louvor")
    elif nota >= 8:
        print("O aluno foi aprovado com distinção")
    else:
        print("o aluno foi aprovado")
elif nota >= 3.5:
    print("O aluno foi reprovado, mas poderá fazer o exame")
else:
    print("o aluno foi reprovado e não poderá fazer o exame")
```

```
Digite a nota do aluno: 8
O aluno foi aprovado com distinção
```

7.4. Comando FOR

[CF] O comando for é um comando de controle que permite que um bloco de comandos seja repetitivamente executado. O número de execuções depende da variável de controle.

[CF] A forma geral do comando for é:

```
for v in range(vi, vf, incr):
```

sendo que:

- "v" é a variável do controle do laço for.
- "vi" é o valor inicial de v.
- "vf" é o valor final ou máximo de v.
- "incr" é o incremento de v. Se for omitido, é suposto como de valor 1.

[CF] O for é finalizado com o final da identação, assim como no caso do if.

[E] Exemplo:

```
for v in range(0, 10):
    bloco de comandos
```

7.4.1. Laços de FOR encaixados

[C] É possível encaixar dois ou mais comandos for.

[C] Para as estruturas FOR encaixadas, sua organização interna é ditada pela identação do código.

7.5 Comando WHILE

A estrutura WHILE (while loop) executa o laço enquanto uma expressão lógica/relacional for verdadeira:

```
do while (var == 0):
    bloco de comandos
```

7.6. Comando BREAK

[CF] O comando break termina abruptamente um laço for, direcionando o fluxo do programa para a primeira linha de comando após aquele respectivo comando for envolvido.

7.7. Comando CONTINUE

[CF] O comando continue é um comando executável que somente passa o controle para o próximo comando executável. A forma geral é:

```
continue
```

Capítulo 8: Comandos de Entrada/Saída

8.1. Introdução

[D] Os "Comandos de Entrada" fornecem um método de transferir dados de um dispositivo periférico (como um teclado) ou de um arquivo interno para a memória principal. Este processo é chamado de "leitura de dados".

[D] Os "Comandos de Saída" fornecem um meio de transferir dados da memória principal para um dispositivo periférico (como um monitor de vídeo ou impressora) ou um arquivo interno. Este processo é chamado de "escrita" ou "gravação" de dados.

[C] Alguns comandos de E/S permitem que se edite os dados durante a sua transferência.

[D] Neste capítulo somente serão apresentados os comandos de E/S que transferem dados (open, read, write). Comandos auxiliares não serão discutidos.

8.2. Registros, Arquivos e Unidades

[D] Um "caractere" é um único símbolo, tal como a letra z, o dígito 4 ou um asterisco *.

[D] Um "campo" é uma sucessão de caracteres, que representa alguma informação, tal como ANTONIO ou 128.47.

[D] Um "registro" é um grupo de campos que reúne informações sobre um único item, tal como:

ANTONIO 26-01-1937 6543.21
(nome) (data nasc.) (salário)

[D] Um "arquivo" é um grupo de registros, tal como:

ANTONIO 26-01-1937654 3.21
ARNALDO 22-10-1945 987.65
M.CRISTINA07-09-1942123 4.56

Temos acima um arquivo, com três registros, cada registro com três campos e comprimento total de 27 caracteres.

[C] Cada comando de E/S opera sobre um único registro.

[D] Uma unidade de E/S é um meio de se referir a um arquivo. Usa-se números de unidades de E/S para distinguir um arquivo de outro.

[C] O comando OPEN estabelece a conexão a um particular arquivo.

8.3 Comando OPEN

[C] O comando OPEN serve para inicializar um arquivo.

[CF] A forma geral simplificada do comando OPEN é:

```
open('arquivo.csv', 'r')
```

sendo que:

- $r \rightarrow$ Ler
- $w \rightarrow$ Escrever
- $a \rightarrow$ Adicionar

8.4. Comando CLOSE

[D] O comando CLOSE serve para finalizar um arquivo. É o oposto de OPEN.

[CF] A forma geral simplificada do comando CLOSE é:

```
arquivo.close()
```

8.5. Comando READ

[C] A fim de fazer a leitura de um arquivo, primeiro precisamos abri-lo, utilizando o comando open(), como indicado acima.

[D] Os comandos READ transferem dados de entrada para a memória de trabalho.

[CF] A forma geral simplificada do comando read() é:

```
arquivo.read()
```

8.6. Comando PRINT

[D] O comando print() escreve na tela o conjunto de informações desejadas.

In [2]:

```
print("Olá mundo")
```

```
Olá mundo
```

In [3]:

```
variavel = "Olá Mundo"
print(variavel)
```

```
Olá Mundo
```

Capítulo 9: Comando FORMAT e Especificações de Formato

9.1. Introdução

[D] FORMAT é o comando que contém uma lista de especificações de formato que permite ao programador ter controle sobre os dados de entrada e saída.

[E] Exemplo do uso de um comando de formatação:

In [22]:

```
print("Meu nome é {}, Eu peso {:.2f} Kg".format("Felipe", 72))
```

```
Meu nome é Felipe, Eu peso 72.00 Kg
```

[C] Os valores contidos dentro do format() devem substituir as chaves na expressão acima. Dentro das chaves, pode-se colocar parâmetros para formatar as variáveis como desejado. No exemplo acima, :.2f define que o valor será um float (real) com duas casas após a vírgula.

9.2. Especificações de Formato (EF) de Conversão

[C] Para ter acesso à todos os formatos de conversão possíveis, acesse a documentação da função format(). https://python-reference.readthedocs.io/en/latest/docs/str/format.html (https://python-reference.readthedocs.io/en/latest/docs/str/format.html)

In [33]:

```
var1 = 111
var2 = 222.22
var3 = 3
var4 = 44.44
print("binário de var1 = {:b}, var2 = {:.1f}, +var3 = {:+}, %var4 = {:%}".format(var1, var2
```

```
binário de var1 = 1101111, var2 = 222.2, +var3 = +3, %var4 = 4444.000000%
```

9.3. Especificações de Espaçamento

[D] As especificações de edição mais usadas no Python são:

Especificação	Função
\t	espaçamento (pular coluna excel)
\n	nova linha

[E] Exemplos:

In [38]:

```
print("Olá!\n tudo bem?")
```

```
Olá!
 tudo bem?
```

In [37]:

```
print("Olá! \t tudo bem?")
```

```
Olá!     tudo bem?
```

Capítulo 10: Funções

10.1. Introdução

[D] Um "subprograma", como o próprio nome diz, é uma sequência de comandos que fazem parte de um programa.

[C] Um programa executável consiste num único programa principal e subprogramas opcionais.

[D] Uma "função" é um tipo de subprograma que executa uma determinada tarefa, e pode ser chamada quando conveniente.

10.2. Argumentos de Subprogramas

[D] Um "argumento de subprograma" é uma entidade que transfere um valor para ou de um subprograma.

[D] Há duas espécies de argumentos: atuais e mudos.

[D] Os argumentos "atuais" são escritos no comando que chama o subprograma.

[D] Os argumentos "mudos" são especificados na definição do subprograma.

10.3. Funções Intrínsecas (FI)

[C] As funções intrínsecas são procedimentos tão usados que achou-se por bem inclui-las como parte integrante do Python, e portanto, já vem definidas.

[D] Uma FI sempre é usada em uma expressão e retorna o valor calculado da função para o argumento, que se encontra entre parênteses:

```
fi(argumento)
```

[E] Exemplos: y = len(x) , FI = len , Função: Mostra o comprimento do argumento; argumento = x. y = len(x**2) , FI = len , nome: Mostra o comprimento do argumento; argumento = x**2.

[C] A tabela abaixo contém algumas das funções intrínsecas mais usadas em Python.

FUNÇÃO	DESCRIÇÃO	RESULTADO
int(x)	converte para o tipo inteiro, truncando a mantissa	inteiro
real(x)	converte para o tipo real	real
str(x)	converte numérico para string	string
abs(x)	valor absoluto (ou módulo)	mantém

max(x,z,w,...)	encontra o maior valor entre os argumentos	mantém
min(x,z,w,...)	encontra o menor valor entre os argumentos	mantém
len(x)	comprimento de uma variável ou constante caractere	inteiro
lista.index(x,z)	localiza a variável ou constante caractere x dentro da variável ou constante z	inteiro

10.4. Funções DEF

[D] Um subprograma função DEF é usado do mesmo modo que se usa uma função intrínseca. A diferença é que o subprograma DEF é escrito pelo próprio programador. Isto serve para resolver problemas como, por exemplo, para se ter uma função que obtenha o valor de y(x)=x2+3x+2. A função y(x) não teria que ser reescrita diversas vezes ao longo do programa, evitando-se erros.

[D] O subprograma DEF sempre começa com o comando def, seguido por uma sequência de comandos, terminando com o final da identação. O controle de execução pode retornar algum resultado para o comando principal através do comando return.

[D] Regras para o uso de subprogramas def:

1. Deve haver concordância entre os argumentos passados e recebidos, no programa principal e no subprograma.
2. Para ter acesso ao valor retornado por uma função deve-se salvar o seu resultado em uma variável.

10.4.1. Comando DEF

[CF] A forma geral do comando DEF é:

```
def nome(a):
    codigoDaFuncao
    return valor
```

sendo que:

- "nome" é o nome da função e obedece às regras para os nomes das variáveis.
- "a" é um argumento que é passado para ser processado pela função.
- a função return só é utilizada caso seja necessário armazenar algum valor na função em questão.

[CF] A forma de chamar uma função em Python é a seguinte:

```
variavel = nome(b)
```

sendo que:

-"nome" é o nome da função criada.

- "b" é um argumento.

[E] Exemplo.

In [17]:

```
def volume(r, h):
    volume = 3.1415*(r**2)*h
    return volume

r = float(input("Digite o valor do raio em metros: "))
h = float(input("Digite a altura do cilindro em metros: "))
print(volume(r, h))
```

```
Digite o valor do raio em metros: 4
Digite a altura do cilindro em metros: 5
251.32000000000002
```

10.5. Importando Funções

[C] Além das funções intrínsecas do Python e das criadas pelo próprio programador, o mesmo pode se beneficiar de bibliotecas de funções já criadas por outros programadores.

[C] Para tal, em Python utiliza-se a função import.

[C] Desde que as funções estejam instaladas no seu ambiente de desenvolvimento, o Python será capaz de importar estas bibliotecas. Caso não estejam, o download das bibliotecas pode ser feito utilizando os comandos $pip\ install$ " $biblioteca$ " no terminal (prompt de comando).

[C] As bibliotecas podem ter as suas "chamadas" encurtadas utilizando o comando as a seguir. Ele nada mais faz do que renomear as funções importadas. Veja os exemplos abaixo para entender melhor.

[C] A quantidade de bibliotecas disponíveis, assim como o grande número de tarefas as quais estas realizam é um dos grandes trunfos da linguagem Python, e um dos principais motivos para a sua popularização.

[C] Algumas bibliotecas famosas são o Numpy e Scipy (Utilizadas para ciência e matemática), Pandas (Para lidar com dados), Matplotlib (para plotar gráficos), Pygame (para o desenvolvimento de jogos), etc.

[E] Observe alguns exemplos a seguir:

Importando cosseno, e o valor de π

In [22]:

```
import numpy
print(numpy.cos(numpy.pi))
```

```
-1.0
```

In [23]:

```
import numpy as np
print(np.cos(np.pi))
```

```
-1.0
```

Importando uma função para raiz quadrada.

In [24]:

```
print(np.sqrt(4))
```

Out[24]:

```
2.0
```