# Algoritmos e Programação I: Aula 02.*

Faculdade de Computação
Universidade Federal de Mato Grosso do Sul
79070-900 Campo Grande, MS
`http://ava.ufms.br`

## Sumário



# 1 Configurando seu Ambiente de Trabalho

Nesta disciplina utilizaremos a linguagem de programação Python. O Python pode ser obtido em Python ou outras distribuições como a Anaconda Anaconda. Para editar um programa em Python podemos usar o IDLE (instalado junto com o Python), o Spyder (Anaconda) ou qualquer outro editor disponível no ambiente operacional que você utiliza. O spyder também pode ser instalado de forma independente. Para dispositivos móveis, existem também vários aplicativos para programação na linguagem Python e também ambientes na nuvem que possibilitam o desenvolvimento de programa em Python tanto para dispositivos móveis como para desktops e notebooks.

No livro Introdução à Computação com Python: um curso interativo dos professores J.C. de Pina Jr. e C.H. Morimoto tem um tutorial de como instalar o Anaconda (Spyder). A configuração do Python com o Idle é semelhante.

Com o ambiente de trabalho configurado, passamos a edição, compilação/interpretação e execução de programas.

# 2 Editando, Compilando/Interpretando e Executando Programas

O hardware somente entende um programa se ele estiver codificado em linguagem de máquina.

*Este material é para o uso exclusivo da disciplina de Algoritmos e Programação I da FACOM/UFMS e utiliza as referências bibliográficas da disciplina

**Linguagem de máquina** é um conjunto de instruções que é executado diretamente pela Unidade Central de Processamento do Computador (CPU). Cada instrução executa uma tarefa muito específica, tais como `load`, `jump`, ou uma operação na Unidade Lógica Aritmética. Todo o programa executado diretamente na CPU é composto por uma série dessas instruções.

Inicialmente, a única forma de programar um computador era com linguagem de máquina. Logo ficou evidente a dificuldade de utilizar um computador como uma *máquina* de propósito geral. Essa dificuldade motivou a criação de linguagens simbólicas. No início dos anos 50, Grace Hopper desenvolveu o conceito de um programa de computador (compilador A-0 - Arithmetic Language version 0) para converter programas em linguagem de máquina. Os primeiros compiladores usavam uma linguagem com símbolos ou mnemônicas para representar as instruções da linguagem de máquina. Essas linguagens foram denominadas de **Linguagens Simbólicas**.

Nas transparências da aula prática tem um exemplo de programas em linguagem de máquina e linguagem simbólica que multiplicam dois números inteiros.

Mesmo com as linguagens simbólicas a programação dos computadores era muito complexa e os programas eram dependentes do hardware. No final dos anos 50, a equipe liderada por John W. Backus na IBM apresentou a linguagem FORTRAN, que acredita-se que tenha sido o primeiro compilador completo. Além do FORTRAN também surgiram o ALGOL e o COBOL. Essas linguagens eram independentes do hardware e foram denominadas linguagens de alto nível.

O conjunto de linguagens de alto nível é muito grande e neste nosso curso abordaremos a linguagem Python 3.

## 2.1 Programando em Python

Vamos agora descrever como criar, editar, interpretar e executar um programa na linguagem Python. Dado um problema, o primeiro passo e projetar um algoritmo que solucione o problema. Após a solução algorítmica, passamos para a fase da codificação (edição) do programa que implementará o algoritmo na linguagem Python.

Na **codificação**, utilizamos um **editor de texto** ou uma **IDE** para editar o programa. Esses softwares possibilitam ao programador uma série de informações, verificação de sintaxe, etc. Após a codificação o programa deve ser salvo num dispositivo de armazenamento usando um nome de arquivo válido e usando a terminologia da linguagem Python (`nome_programa.py`). Esse arquivo será a entrada para o interpretador Python e é denominado **programa fonte**.

Para que o código no arquivo programa fonte possa ser executado por um computador ele deve ser traduzido para a linguagem de máquina. Essa tarefa é feita por um **interpretador**. Um interpretador Python executa uma série de tarefas, tais como verificação da sintaxe do programa, otimização do código, alocação de registradores, etc., muitas das quais não serão tratadas neste curso. Vamos nos ater nos passos básicos que o interpretador fará para interpretar e executar um programa.

O interpretador Python analisa linha a linha do programa. O Python é uma linguagem multi-paradigma que usa tipos dinâmicos. A cada linha interpretada, a sintaxe e a semântica das instruções é verificada. Se a linha contiver algum erro, o programa é interrompido. Após o erro ser corrigido, o interpretador passa para a linha seguinte, até que todas as linhas sejam "executadas".

Como o Python é uma linguagem multi-paradigma, a composição de um programa Python depende do paradigma que estiver sendo utilizado.

Python é formado por um conjunto de funções (métodos). Algumas dessas funções

fazem parte das bibliotecas existentes na linguagem Python (p. ex. entrada e saída, funções matemáticas, etc.).

Nesse momento nosso programa está pronto para execução. A forma da execução depende do sistema operacional que estamos utilizando. O comando de execução carrega o programa na memória principal e a unidade de controle da CPU inicia a execução do programa. Nesse momento o seu programa toma o controle do seu computador.

## 2.2 Estrutura de um Programa Python

A linguagem Python é multi-paradigma e possui tipos dinâmicos, e foi projetada com o intuito de ser mais simples que as demais linguagens.

Similar ao C e o Java que usam bibliotecas e classes, o Python utiliza módulos (modules) e pacotes (packages) que são conjuntos de módulos. No caso de entrada e saída, não é necessário a utilização de módulos.

```
import math
```

A instrução acima possibilita utilizarmos várias funções que auxiliam na solução de problemas que envolvem funções matemáticas.

O nome de um programa Python é o nome ao programa com a extensão `.py`. O programa pode conter uma ou mais funções, módulos ou *packages*.

Como a linguagem Python é multi-paradigma, um programa nessa linguagem tem uma estrutura bastante flexível. A sintaxe não requer nenhuma ordenação específica para as funções ou classes dentro de um módulo. O programa pode ser composto de um conjunto de funções, módulos, etc. sendo um deles denominado *driver* principal.

A seção das instruções contém o conjunto de instruções que é reconhecida pela linguagem Python e que descreverá a solução do problema que está sendo solucionado.

```
# Declaração dos módulos/packages
# Outros métodos/funções do programa
# Declarações locais
# Instruções
# fim do driver principal
```

## 2.3 Nosso primeiro programa em Python

Descreveremos agora como editar, compilar/interpretar e executar um programa na linguagem Python usando o editor Idle.

Você pode usar o ambiente Windows ou Linux do Laboratório. Faça o login na sua conta, escolha um editor ou um ambiente de programação (IDE), edite, compile e execute o programa abaixo.

Se você escolher o ambiente Linux e não estiver familiarizado com o Linux, faça uma leitura prévia do material referente ao ambiente Unix do Prof. Fábio Henrique Martinez Viduani que está disponível na aula 01.

Você pode usar qualquer editor como `Gedit`, `Emacs`, `Kate`, etc. ou qualquer ambiente de programação (IDE) como `IDLE` ou `Eclipse`. A Figura 1 ilustra a utilização do IDLE no Windows.

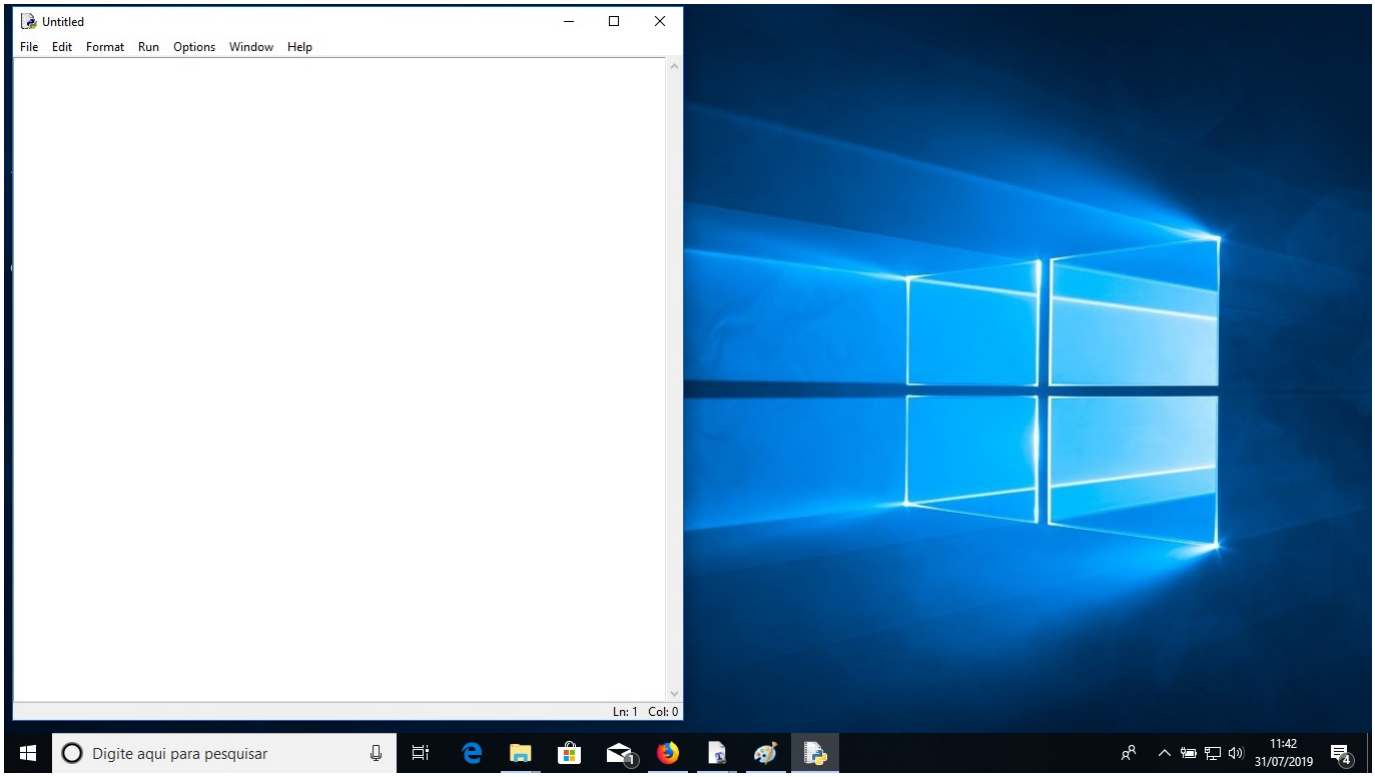


Figura 1: IDLE

O nosso primeiro programa em Python também será o `oi.py`. A instrução `print` (função no Python 3) imprimirá no dispositivo padrão (console) a mensagem que estiver entre aspas simples (' '). Após a impressão do texto, a instrução muda de linha.

Note que as instruções **NÃO** são seguidas de um ";".

```
1 # -*- coding: utf-8 -*-
2 #  Programa: oi.py
3 #  Programador:
4 #  Data: 11/04/2013
5 #  Este programa imprime Oi na tela do computador
6
7 print('Oi!')
8
9 # fim do programa
```

Escolha o ambiente que você achar mais adequado e edite o programa `oi.py`. Esse programa imprime a mensagem `Oi!` na tela do computador. Edite cuidadosamente o programa observado letras maiúsculas e minúsculas. **Não utilize CTRL-C CTRL-V**. Não se preocupe com a sintaxe do programa, pois os detalhes serão explicados no decorrer das aulas.

Diferentemente do C e Java, para esse programa, o Python não necessita importar nenhum módulo adicional. A estrutura de um programa Python é diferente. Posteriormente isso será melhor detalhado.

Para esse programa, o Python não precisa importar nenhum módulo adicional, funções/métodos ou classes.

Todas as linhas que inciam com#, com exceção da primeira, serão ignoradas pelo interpretador e servem para descrever (documentar) o que o programa e cada uma das instruções fazem. A linha

```
1 # -*- coding: utf-8 -*-
```

indica que a codificação do programa será em UTF-8 e evita que o interpretador tenha problemas com caracteres acentuados ou ç dos comentários.

A instrução

```
print('Oi!');
```

indica que o texto (`string`) ‘‘`Oi!`’’ será impresso no dispositivo padrão de saída do computador (no caso a tela do computador).

Se você estiver usando o IDLE, você deve ter uma tela parecida com a Figura 2:

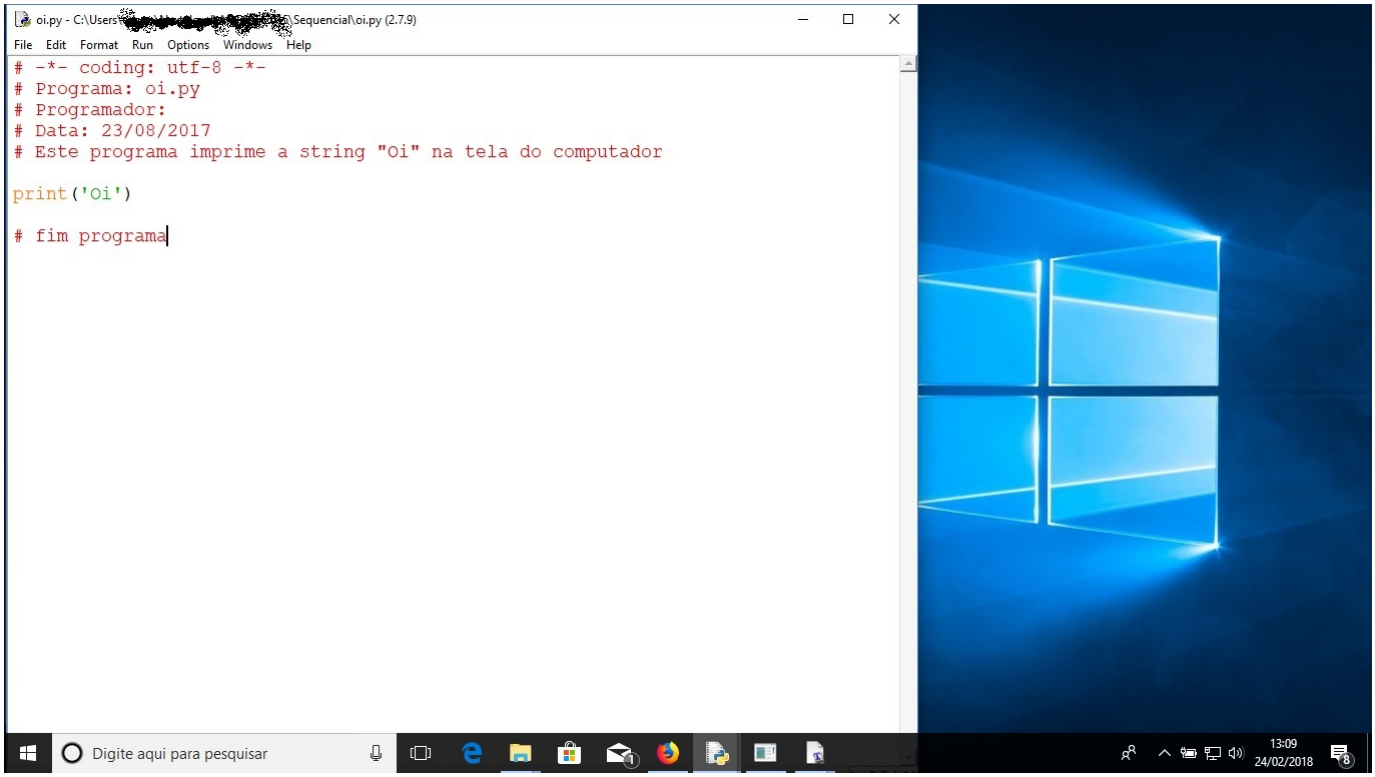


Figura 2: IDLE `oi.py`.

Verifique o diretório que você está e salve o seu programa (use o nome `oi.py`). Se você estiver usando o IDLE, a barra superior indica onde o programa foi salvo.

Depois que você editou o programa, o próximo passo será a compilação/interpretação do programa para gerar um código que o computador “entenda” para executar o programa. Temos duas possibilidades: usar um terminal para compilar ou usar uma IDE.

Primeiro, como descrevemos acima, inicie um terminal e vá para o diretório onde você salvou o programa (arquivo `oi.py`). Se você estiver no Linux, você pode usar o comando do Unix `pwd` para ver que diretório você está e usando. O comando `ls` listará todos os arquivos (e diretórios) do diretório corrente e o comando `cd` permite você mudar de diretórios. Após localizar o diretório onde o arquivo `oi.py` está armazenado, interprete/execute o programa com o seguinte comando:

```
$ python oi.py
```

No caso do Python, o interpretador "executa" o programa instrução a instrução. Para que esse comando funcione corretamente, todos os caminhos para a utilização do interpretador Python devem estar devidamente atribuídos. A Figura 3 ilustra como executar um programa `Python` usando linha de comando.

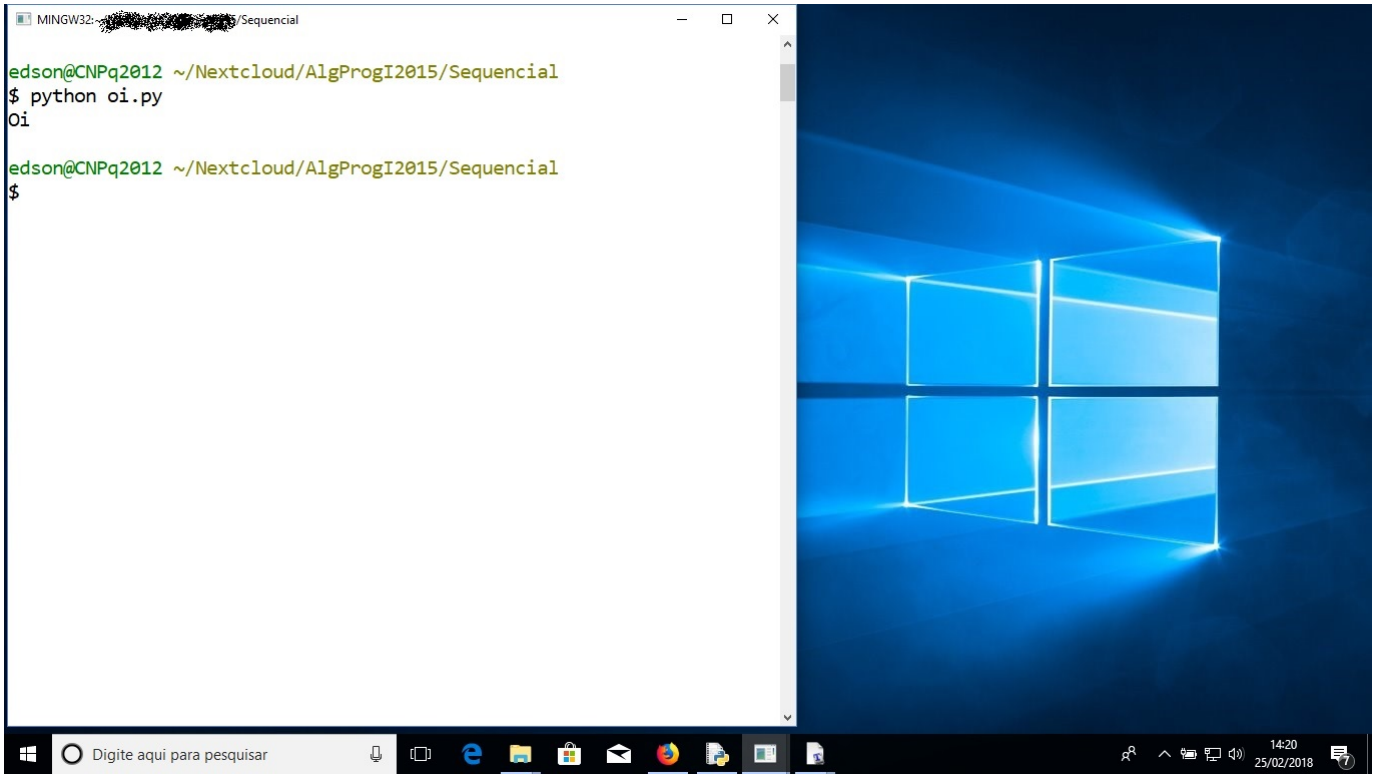


Figura 3: Executando `oi.py`.

Se você editou o programa corretamente a execução do comando de interpretação não gerará nenhuma advertência ou erro. Caso isso ocorra, veja o número da linha e verifique o que você digitou de forma errada.

O resultado será a impressão da string "Oi!" na tela do computador:

```
$ Oi!
```

## 2.4 Editando, Compilando/Interpretando o programa `soma.py`

Anteriormente apresentamos o algoritmo e o programa ler dois números inteiros, calcular a sua soma e imprimir o resultado usando um formato específico. Agora vamos analisar com mais detalhes o problema da soma de dois números inteiros.

A entrada é dada por dois números inteiros e a saída deve imprimir o primeiro número inteiro lido, seguido de um espaço, o sinal de `+`, um espaço, o segundo número inteiro lido, seguido de um espaço, o sinal de `=`, um espaço e o valor da soma. Após a impressão do resultado, imprimir uma nova linha, de acordo com os exemplos abaixo:

```
5
6
```

```
5 + 6 = 11
```

```
-55
55
```

```
-55 + 55 = 0
```

O programa `soma.py` implementa uma solução na linguagem `Python` para o algoritmo. Usando um editor (ou um ambiente de programação) edite o programa `soma.py` abaixo:

```
1  # -*- coding: utf-8 -*-
2  # Programa: soma.py
3  # Programador:
4  # Data: 23/08/2017
5  # Este programa lê dois números inteiros, calcula a soma dos números
6  # e imprime a soma calculada
7  # início do programa
8  # descrição das variáveis utilizadas
9  # int numero1, numero2, soma
10 # pré: numero1 numero2
11 # Passo 1. Leia dois números inteiros
12 numero1 = int(input())
13 numero2 = int(input())
14 # Passo 2. Calcule a soma dos dois números
15 soma = numero1 + numero2
16 # Passo 3. Imprima a soma dos números
17 print('{0:d} + {1:d} = {2:d}'.format(numero1, numero2, soma))
18 # pós: soma and soma == numero1 + numero2
19 # fim do programa
```

Como observamos acima, a linha

```
# -*- coding: utf-8 -*-
```

define que os caracteres do programa utilizarão a codificação UTF-8. Caso essa instrução não seja colocada, você pode ter problemas com caracteres com acentuação ou ç nos comentários do seu programa.

Todos as demais linhas que iniciam com `#` serão ignoradas pela interpretador, além disso, o Python utiliza tipagem dinâmica, em função disso, o programa não tem declaração de variáveis. Posteriormente esse tópico será tratado com mais detalhes. Com relação as instruções, o Python não utiliza "`;`" para indicar o final de uma instrução, nem "`{,}`" para separar blocos de instruções. Os blocos são definidos pelo nível de indentação das instruções.

A instrução

```
numero1 = int(input())
```

indica que um fluxo de entrada será lido da entrada padrão e esse fluxo será "convertido" num número inteiro e atribuído à variável `numero1`.

A instrução

```
print('{0:d} + {1:d} = {2:d}'.format(numero1, numero2, soma))
```

imprime a string '{0:d} + {1:d} = {2:d'} no dispositivo padrão do computador, substituindo {0:d} com o valor da variável numero1 armazenado na memória, {1:d} com o valor da variável numero2 armazenado na memória e {2:d} com o valor armazenado na memória da variável soma. O print no Python sempre imprime a saída numa nova linha, a não ser que uma instrução seja usando na função print para permanecer na mesma linha.

Se você estiver usando o Idle, você também pode usar a opção de execução disponível na IDE.

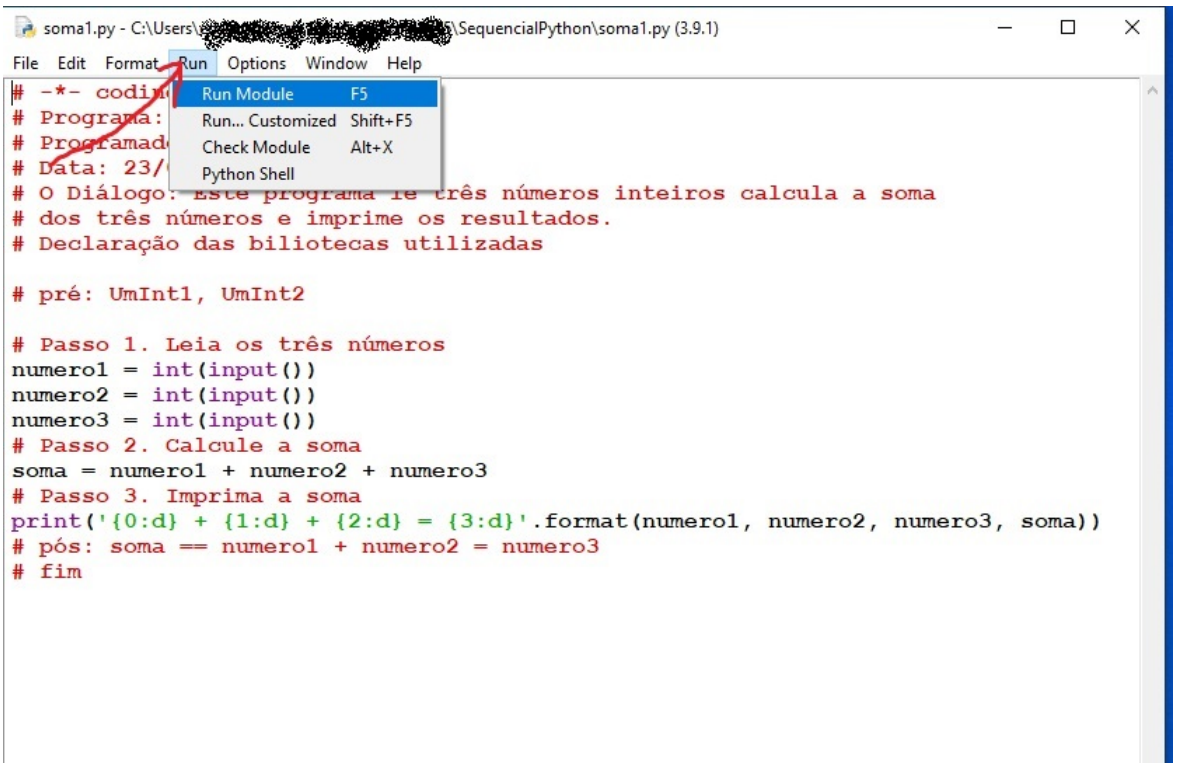


Figura 4: Executando Executando um Programa no IDLE.

# 3 Exercícios

1. Leia as transparências da Aula 03, o vídeo do programa oi.py e projete e implemente um programa na linguagem Python que imprima a mensagem "Oi UFMS!". Usando o ambiente de programação VPL implemente o programa oiufms.py. Interprete/Execute e avalie a sua solução.

2. Leia o material das Aulas 01 e 02, analise o programa soma.py acima, e modifique-o para projetar e implementar um programa em Python para ler três números inteiros, calcular a sua soma e imprimir o resultado usando um formato específico. Salve seu programa com o nome soma1.py.

   A entrada é dada por três números inteiros e a saída deve imprimir o primeiro número inteiro lido, seguido de um espaço, o sinal de +, seguido de um espaço, o segundo número inteiro lido, seguido de um espaço, o sinal de +, seguido de um espaço, o terceiro número inteiro lido, seguido de um espaço, o sinal de =, um espaço e o valor

da soma. Após a impressão do resultado, imprimir uma nova linha, de acordo com os exemplos abaixo:

```
5
6
7
```

```
5 + 6 + 7 = 18

```

```
-55
22
33
```

```
-55 + 22 + 33 = 0

```

Use o ambiente de programação VPL e o esboço do programa `soma1.py`. Compile/interprete, execute e avalie a sua solução.