BCM0505-22 – Processamento da Informação

Entrada e Saída

Maycon Sambinelli
m.sambinelli@ufabc.edu.br
http://professor.ufabc.edu.br/~m.sambinelli/

Outline

Entrada e Saída

I/O Com Python

Redirecionando a stdin e stdout

Argumentos na Linha de Comando

Stddraw

Exercícios

Entrada e Saída

Introdução

Em computação, costumamos categorizar alguns dispositivos como sendo de *entrada* ou *saída*

- Um dispositivos de **entrada** é um dispositivo que pode ser usado para enviar dados e comandos ao computador
  - **Ex:** Teclado, Webcam, Microfone, Mouse, Tela Touch Screen
- Um dispositivo de **saída** é um dispositivo capaz de exibir as informações processadas pelo computador
  - **Ex:** Monitor, Impressora, Caixa de Som, Projetor
- A expressão Entrada e Saída é geralmente abreviada para **I/O** (do inglês *input/output*)

Entrada e Saída Padrão

Os sistemas computacionais atuais trabalham com um conceito chamado **entrada e saída padrão** que cria uma abstração dos dispositivos de **I/O**

- A ***Entrada Padrão***, ou ***stdin*** (de *standard input*), fornece o *stream* de caracteres que será lido pelo programa

Entrada e Saída Padrão

Os sistemas computacionais atuais trabalham com um conceito chamado **entrada e saída padrão** que cria uma abstração dos dispositivos de **I/O**

- A ***Entrada Padrão***, ou ***stdin*** (de *standard input*), fornece o *stream* de caracteres que será lido pelo programa
    - Por padrão, a *stdin* está associada ao teclado

Entrada e Saída Padrão

Os sistemas computacionais atuais trabalham com um conceito chamado **entrada e saída padrão** que cria uma abstração dos dispositivos de **I/O**

- A ***Entrada Padrão***, ou ***stdin*** (de *standard input*), fornece o *stream* de caracteres que será lido pelo programa
  - Por padrão, a *stdin* está associada ao teclado
- A ***Saída Padrão***, ou ***stdout*** (de *standard output*), exibe o *stream* de caracteres que são enviador para impressão

Entrada e Saída Padrão

Os sistemas computacionais atuais trabalham com um conceito chamado **entrada e saída padrão** que cria uma abstração dos dispositivos de **I/O**

- A ***Entrada Padrão***, ou ***stdin*** (de *standard input*), fornece o *stream* de caracteres que será lido pelo programa
  - Por padrão, a *stdin* está associada ao teclado
- A ***Saída Padrão***, ou ***stdout*** (de *standard output*), exibe o *stream* de caracteres que são enviador para impressão
  - Por padrão, a *stdout* está associada a saída do terminal que roda o nosso programa

Entrada e Saída Padrão

Os sistemas computacionais atuais trabalham com um conceito chamado **entrada e saída padrão** que cria uma abstração dos dispositivos de **I/O**

- A ***Entrada Padrão***, ou ***stdin*** (de *standard input*), fornece o *stream* de caracteres que será lido pelo programa
  - Por padrão, a *stdin* está associada ao teclado
- A ***Saída Padrão***, ou ***stdout*** (de *standard output*), exibe o *stream* de caracteres que são enviador para impressão
  - Por padrão, a *stdout* está associada a saída do terminal que roda o nosso programa

Entrada e Saída Padrão

Os sistemas computacionais atuais trabalham com um conceito chamado **entrada e saída padrão** que cria uma abstração dos dispositivos de **I/O**

- A ***Entrada Padrão***, ou ***stdin*** (de *standard input*), fornece o *stream* de caracteres que será lido pelo programa
  - Por padrão, a *stdin* está associada ao teclado
- A ***Saída Padrão***, ou ***stdout*** (de *standard output*), exibe o *stream* de caracteres que são enviador para impressão
  - Por padrão, a *stdout* está associada a saída do terminal que roda o nosso programa

Essa abstração permite que possamos ler e escrever em outros dispositivos sem a necessidade de modificar o nosso programa

- Basta trocar o dispositivo associado à *stdin* e/ou *stdout*
- Para o nosso programa é transparente
  - Ele lê uma *stream* de caracteres de *stdin*
  - Ele escreve uma *stream* de caracteres na *stdout*

I/O Com Python

Saída: `print()`

Usamos a função `print()` para escrever um valor na *stdout*

- A função `print` pode receber zero ou mais argumentos:

Saída: `print()`

Usamos a função `print()` para escrever um valor na *stdout*

- A função `print` pode receber zero ou mais argumentos:
  - `print(x)` imprime o valor da expressão `x` e quebra a linha

Saída: `print()`

Usamos a função `print()` para escrever um valor na *stdout*

- A função `print` pode receber zero ou mais argumentos:
  - `print(x)` imprime o valor da expressão `x` e quebra a linha
  - `print(x, y, z)` imprime o valor da expressão `x`, um espaço, o valor da expressão `y`, um espaço, o valor da expressão `z` e quebra a linha

Saída: `print()`

Usamos a função `print()` para escrever um valor na *stdout*

- A função `print` pode receber zero ou mais argumentos:
  - `print(x)` imprime o valor da expressão `x` e quebra a linha
  - `print(x, y, z)` imprime o valor da expressão `x`, um espaço, o valor da expressão `y`, um espaço, o valor da expressão `z` e quebra a linha
    - Não há limite para o número de parâmetros passados para a função `print`

Saída: `print()`

Usamos a função `print()` para escrever um valor na *stdout*

- A função `print` pode receber zero ou mais argumentos:
  - `print(x)` imprime o valor da expressão `x` e quebra a linha
  - `print(x, y, z)` imprime o valor da expressão `x`, um espaço, o valor da expressão `y`, um espaço, o valor da expressão `z` e quebra a linha
    - Não há limite para o número de parâmetros passados para a função `print`
    - Se quisermos evitar que a linha seja quebrada, devemos escrever `end=''` depois do último parâmetro: `print(x, y, z, end='')`

Saída: `print()`

Usamos a função `print()` para escrever um valor na *stdout*

- A função `print` pode receber zero ou mais argumentos:
  - `print(x)` imprime o valor da expressão `x` e quebra a linha
  - `print(x, y, z)` imprime o valor da expressão `x`, um espaço, o valor da expressão `y`, um espaço, o valor da expressão `z` e quebra a linha
    - Não há limite para o número de parâmetros passados para a função `print`
    - Se quisermos evitar que a linha seja quebrada, devemos escrever `end=''` depois do último parâmetro: `print(x, y, z, end='')`
  - `print()` quebra a linha

Saída: exemplos

Código	Stdout
`print(3)`	`3\n`
`print(43, end='')`	`3`
`print(1/3)`	`0.3333333333\n`
`print(False)`	`False\n`
`print('Eu Amo Python')`	`Eu Amo Python\n`

`x = 54`
`y = 73`
`print(x,y)`
	`54 73\n`

`a = 'Feliz'`
`b = 'Ano'`
`c = 2025`
`print(a,b,c)`
	`Feliz Ano 2025\n`

Help

Você pode acessar a documentação de uma dada função em Python com o auxílio da função `help`

```
>>> help(print)
Help on built-in function print in module builtins:

print(*args, sep=' ', end='\n', file=None, flush=False)
    Prints the values to a stream, or to sys.stdout by default.

    sep
      string inserted between values, default a space.
    end
      string appended after the last value, default a newline.
    file
      a file-like object (stream); defaults to the current sys.stdout.
    flush
      whether to forcibly flush the stream.
```

String de Formatação

Às vezes, queremos fazer um controle mais fino da saída. Python possui um recurso chamado **string de formatação** que amplia os caracteres especiais que podemos utilizar em um literal de string.

String de Formatação

Às vezes, queremos fazer um controle mais fino da saída. Python possui um recurso chamado **string de formatação** que amplia os caracteres especiais que podemos utilizar em um literal de string.

- Para criar um literal de *string de formatação* escrevemos a letra `f` na frente de um literal de `str`
  - Ex: `f"Nome: {nome}, Peso: {peso:.2f}"`
    - Código entre `{}` é avaliado no momento da formatação.
    - `:.2f`: formata números de ponto flutuante com 2 casas decimais.

String de Formatação

Às vezes, queremos fazer um controle mais fino da saída. Python possui um recurso chamado **string de formatação** que amplia os caracteres especiais que podemos utilizar em um literal de string.

- Para criar um literal de *string de formatação* escrevemos a letra `f` na frente de um literal de `str`
  - Ex: `f"Nome: {nome}, Peso: {peso:.2f}"`
    - Código entre `{}` é avaliado no momento da formatação.
    - `:.2f`: formata números de ponto flutuante com 2 casas decimais.

```
nome = "Claudio Silva"
peso = 95.7583
print(f"Nome: {nome}, Peso: {peso:.2f}")
```

Saída:

```
Nome: Claudio Silva, Peso: 95.76
```

Entrada: `input()`

A função `input()` lê uma linha da *stdin* e retorna essa linha como uma `str`

- `input()` lê uma string da *stdin*
- `input(s)` imprime a string `s` na *stdout* e lê uma string da *stdin*

Entrada: `input()`

A função `input()` lê uma linha da *stdin* e retorna essa linha como uma `str`

- `input()` lê uma string da *stdin*
- `input(s)` imprime a string `s` na *stdout* e lê uma string da *stdin*

```
nome = input("Digite o seu nome: ")
print(f"Olá, {nome}")
```

Terminal:

```
Digite o seu nome: Dave
Olá, Dave
```

Entrada: `input()`

```
x = input()
y = input()
print(x + y)
```

- Qual a saída do programa se o usuário digitar 13 e 7?

Entrada: `input()`

```
x = input()
y = input()
print(x + y)
```

- Qual a saída do programa se o usuário digitar `13` e `7`?
- `input()` **sempre** retorna uma `str`

Entrada: `input()`

```
x = input()
y = input()
print(x * y)
```

- Qual a saída do programa se o usuário digitar 13 e 7?

Entrada: `input()`

```
x = input()
y = input()
print(x * y)
```

- Qual a saída do programa se o usuário digitar `13` e `7`?

```
Traceback (most recent call last):
  File "<python-input-2>", line 1, in <module>
    print(x * y)
          ~~^~~
TypeError: can't multiply sequence by non-int of type 'str'
```

Entrada: `input()`

Quando lemos um dado em Python com a função `input()`, é comum convertê-lo imediatamente no tipo desejado

- `int(input())`: lê um inteiro
- `float(input())`: lê um número real

```
x = float(input())
y = float(input())
print(x * y)
```

Salada na sua tela: stdin e stdout

- Por padrão, o terminal exibe cadeias da *stdin* e da *stdout*.
- É importante entender que são duas coisas diferentes, pois um corretor automático só vê a *stdout*

Salada na sua tela: stdin e stdout

- Por padrão, o terminal exibe cadeias da *stdin* e da *stdout*.
- É importante entender que são duas coisas diferentes, pois um corretor automático só vê a *stdout*

```
print("Nota da P1:")
p1 = float(input())

print("Nota da P2:")
p2 = float(input())

print("Nota da P3:")
p3 = float(input())
media = (p1 + 2 * p2 + 3 * p3) / 6.0
print("Média Final: ", media)
```

Salada na sua tela: stdin e stdout

Uma possível execução:

```
$ python media_final.py
Nota da P1:
7.8
Nota da P2:
5.0
Nota da P2:
9.2
Média Final:  7.566666666666666
```

stdin

```
7.8
5.0
9.2
```

stdout

```
Nota da P1:
Nota da P2:
Nota da P2:
Média Final:  7.566666666666666
```

Redirecionando a stdin e stdout

Redirecionamento da stdin e stdout

O programa abaixo é capaz de calcular a sua média final em PI

```
p1 = float(input())
p2 = float(input())
p3 = float(input())
media = (p1 + 2 * p2 + 3 * p3) / 6.0
print(media)
```

Redirecionamento da stdin e stdout

O programa abaixo é capaz de calcular a sua média final em PI

```
p1 = float(input())
p2 = float(input())
p3 = float(input())
media = (p1 + 2 * p2 + 3 * p3) / 6.0
print(media)
```

Digamos que você tenha recebido o seu boletim (`boletim.txt`) escolar e ele contenha as suas notas

```
7.4
8.9
9.2
```

Redirecionando a stdin

O código `<` abaixo redireciona a *stdin* para apontar para o arquivo `boletim.txt`

- Quando o programa `media_final.py` executar a instrução `input()` para ler uma linha, o *stream* de caracteres será retirado do arquivo `boletim.txt`
- Semanticamente, é como seu o usuário estivesse digitando o que está escrito no arquivo

```
$ python media_final.py < boletim.txt
```

Obs: Suporte nativo para macOS, Linux e FreeBSD. No Windows, requer ajustes manuais.

Redirecionando a stdout

O código `>` abaixo redireciona a *stdout* para apontar para o arquivo `media_final.txt`

- Quando o programa `media_final.py` executar a instrução `print()`, ao invés de escrever no terminar, o *stream* de caracteres será escrito no arquivo `media_final.txt`

```
$ python media_final.py > media_final.txt
```

Redirecionando a stdout

O código `>` abaixo redireciona a *stdout* para apontar para o arquivo `media_final.txt`

- Quando o programa `media_final.py` executar a instrução `print()`, ao invés de escrever no terminar, o *stream* de caracteres será escrito no arquivo `media_final.txt`

```
$ python media_final.py > media_final.txt
```

Podemos redireciona a *stdin* e *stdout* ao mesmo tempo

```
$ python media_final.py < boletim.txt > media_final.txt
```

Ligando o stdout de um programa no stdin de outro

```
$ python prog1.py | python prog2.py
```

- A forma mais simples de comunicação entre programas distintos é chamada de piping

Obs: Suporte nativo para macOS, Linux e FreeBSD. No Windows, requer ajustes manuais.

Ligando o stdout de um programa no stdin de outro

```
$ python prog1.py | python prog2.py
```

- A forma mais simples de comunicação entre programas distintos é chamada de piping
- Nela, conectamos a *stdout* de um programa ao *stdin* de outro

Obs: Suporte nativo para macOS, Linux e FreeBSD. No Windows, requer ajustes manuais.

Ligando o stdout de um programa no stdin de outro

```
$ python prog1.py | python prog2.py
```

- A forma mais simples de comunicação entre programas distintos é chamada de piping
- Nela, conectamos a *stdout* de um programa ao *stdin* de outro
- O símbolo `|` conecta a *stdout* de `prog1.py` na *stdin* de `prog2.py`

Obs: Suporte nativo para macOS, Linux e FreeBSD. No Windows, requer ajustes manuais.

Ligando o stdout de um programa no stdin de outro

```
$ python prog1.py | python prog2.py
```

- A forma mais simples de comunicação entre programas distintos é chamada de piping
- Nela, conectamos a *stdout* de um programa ao *stdin* de outro
- O símbolo `|` conecta a *stdout* de `prog1.py` na *stdin* de `prog2.py`
- O sistema operacional Unix introduziu e popularizou esse recurso

Obs: Suporte nativo para macOS, Linux e FreeBSD. No Windows, requer ajustes manuais.

Ligando o stdout de um programa no stdin de outro

```
$ python prog1.py | python prog2.py
```

- A forma mais simples de comunicação entre programas distintos é chamada de piping
- Nela, conectamos a *stdout* de um programa ao *stdin* de outro
- O símbolo `|` conecta a *stdout* de `prog1.py` na *stdin* de `prog2.py`
- O sistema operacional Unix introduziu e popularizou esse recurso
    - Desenvolveu diversas ferramentas simples que faziam muito bem uma coisa muito pequena

Obs: Suporte nativo para macOS, Linux e FreeBSD. No Windows, requer ajustes manuais.

Ligando o stdout de um programa no stdin de outro

```
$ python prog1.py | python prog2.py
```

- A forma mais simples de comunicação entre programas distintos é chamada de piping
- Nela, conectamos a *stdout* de um programa ao *stdin* de outro
- O símbolo `|` conecta a *stdout* de `prog1.py` na *stdin* de `prog2.py`
- O sistema operacional Unix introduziu e popularizou esse recurso
  - Desenvolveu diversas ferramentas simples que faziam muito bem uma coisa muito pequena
  - Combinando essas ferramentas com piping era possível realizar tarefas complexas

Obs: Suporte nativo para macOS, Linux e FreeBSD. No Windows, requer ajustes manuais.

Ligando o stdout de um programa no stdin de outro

```
$ python prog1.py | python prog2.py
```

- A forma mais simples de comunicação entre programas distintos é chamada de piping
- Nela, conectamos a *stdout* de um programa ao *stdin* de outro
- O símbolo `|` conecta a *stdout* de `prog1.py` na *stdin* de `prog2.py`
- O sistema operacional Unix introduziu e popularizou esse recurso
  - Desenvolveu diversas ferramentas simples que faziam muito bem uma coisa muito pequena
  - Combinando essas ferramentas com piping era possível realizar tarefas complexas
  - Essas ferramentas continuam sendo usadas nos sistemas modernos

Obs: Suporte nativo para macOS, Linux e FreeBSD. No Windows, requer ajustes manuais.

Ligando o stdout de um programa no stdin de outro

```
$ python prog1.py | python prog2.py
```

- A forma mais simples de comunicação entre programas distintos é chamada de piping
- Nela, conectamos a *stdout* de um programa ao *stdin* de outro
- O símbolo `|` conecta a *stdout* de `prog1.py` na *stdin* de `prog2.py`
- O sistema operacional Unix introduziu e popularizou esse recurso
  - Desenvolveu diversas ferramentas simples que faziam muito bem uma coisa muito pequena
  - Combinando essas ferramentas com piping era possível realizar tarefas complexas
  - Essas ferramentas continuam sendo usadas nos sistemas modernos
  - Estabeleceu a filosofia KISS (*"Keep It Simple, Stupid"*)

Obs: Suporte nativo para macOS, Linux e FreeBSD. No Windows, requer ajustes manuais.

Um Programa que determina se você passou em PI

```
media_final = float(input())
ans = media_final >= 5
print(f"Aprovou-se: {ans}")
```

Um Programa que determina se você passou em PI

```python
media_final = float(input())
ans = media_final >= 5
print(f"Aprovou-se: {ans}")
```

```
$ python media_final.py | python conceito.py
```

Um Programa que determina se você passou em PI

```python
media_final = float(input())
ans = media_final >= 5
print(f"Aprovou-se: {ans}")
```

```
$ python media_final.py | python conceito.py
```

Semanticamente equivalente ao seguinte código, mas nenhum arquivo é criado

```
$ python media_final.py  > aux.txt
$ python conceito.py < aux.txt
```

Argumentos na Linha de Comando

Passando Argumentos na Linha de Comando

Quando invocamos o nosso programa, podemos passar argumentos via linha de comando

```
$ python stock_trade.py 35 ITUB3 89.90 100.34
ITUB3
====================
Cost: 1171.0
Price: 1562.5
Spread: 391.5 (33.43 %)
```

Passando Argumentos na Linha de Comando

Quando invocamos o nosso programa, podemos passar argumentos via linha de comando

```
$ python stock_trade.py 35 ITUB3 89.90 100.34
ITUB3
====================
Cost: 1171.0
Price: 1562.5
Spread: 391.5 (33.43 %)
```

Como lembrar de tudo isso?

- Programas profissionais geralmente suportam o parâmetro `--help` e respondem ao comando `man programa`

Passando Argumentos na Linha de Comando

```
$ python meuprograma.py arg1 arg2 arg3
```

- Podemos passar quantos argumentos quisermos

Passando Argumentos na Linha de Comando

```
$ python meuprograma.py arg1 arg2 arg3
```

- Podemos passar quantos argumentos quisermos
- Para acessarmos os argumentos dentro do nosso programa, precisamos adicionar a linha `import sys`

Passando Argumentos na Linha de Comando

```
$ python meuprograma.py arg1 arg2 arg3
```

- Podemos passar quantos argumentos quisermos
- Para acessarmos os argumentos dentro do nosso programa, precisamos adicionar a linha `import sys`
  - O comando `import xxx` habilita o seu programa a usar funcionalidades implementadas na biblioteca `xxx`

Passando Argumentos na Linha de Comando

```
$ python meuprograma.py arg1 arg2 arg3
```

- Podemos passar quantos argumentos quisermos
- Para acessarmos os argumentos dentro do nosso programa, precisamos adicionar a linha `import sys`
  - O comando `import xxx` habilita o seu programa a usar funcionalidades implementadas na biblioteca `xxx`
  - Existem bibliotecas fornecidas pelos próprios criadores da linguagem Python (*biblioteca padrão*) e existem bibliotecas feitas por terceiros (empresas/hobbystas/você!)

Passando Argumentos na Linha de Comando

```
$ python meuprograma.py arg1 arg2 arg3
```

- Podemos passar quantos argumentos quisermos
- Para acessarmos os argumentos dentro do nosso programa, precisamos adicionar a linha `import sys`
  - O comando `import xxx` habilita o seu programa a usar funcionalidades implementadas na biblioteca `xxx`
  - Existem bibliotecas fornecidas pelos próprios criadores da linguagem Python (*biblioteca padrão*) e existem bibliotecas feitas por terceiros (empresas/hobbystas/você!)
  - Criamos bibliotecas com funcionalidades úteis que podem ser utilizadas em diversos programas

Passando Argumentos na Linha de Comando

```
$ python meuprograma.py arg1 arg2 arg3
```

- Podemos passar quantos argumentos quisermos
- Para acessarmos os argumentos dentro do nosso programa, precisamos adicionar a linha `import sys`
  - O comando `import xxx` habilita o seu programa a usar funcionalidades implementadas na biblioteca `xxx`
  - Existem bibliotecas fornecidas pelos próprios criadores da linguagem Python (*biblioteca padrão*) e existem bibliotecas feitas por terceiros (empresas/hobbystas/você!)
  - Criamos bibliotecas com funcionalidades úteis que podem ser utilizadas em diversos programas
- Podemos acessar o i-ésimo argumento fornecido na linha de comando por `sys.argv[i]`

Passando Argumentos na Linha de Comando

```
$ python meuprograma.py arg1 arg2 arg3
```

- Podemos passar quantos argumentos quisermos
- Para acessarmos os argumentos dentro do nosso programa, precisamos adicionar a linha `import sys`
  - O comando `import xxx` habilita o seu programa a usar funcionalidades implementadas na biblioteca `xxx`
  - Existem bibliotecas fornecidas pelos próprios criadores da linguagem Python (*biblioteca padrão*) e existem bibliotecas feitas por terceiros (empresas/hobbystas/você!)
  - Criamos bibliotecas com funcionalidades úteis que podem ser utilizadas em diversos programas
- Podemos acessar o i-ésimo argumento fornecido na linha de comando por `sys.argv[i]`
  - O tipo do argumento é sempre `str`

Passando Argumentos na Linha de Comando

```
$ python meuprograma.py arg1 arg2 arg3
```

- Podemos passar quantos argumentos quisermos
- Para acessarmos os argumentos dentro do nosso programa, precisamos adicionar a linha `import sys`
  - O comando `import xxx` habilita o seu programa a usar funcionalidades implementadas na biblioteca `xxx`
  - Existem bibliotecas fornecidas pelos próprios criadores da linguagem Python (*biblioteca padrão*) e existem bibliotecas feitas por terceiros (empresas/hobbystas/você!)
  - Criamos bibliotecas com funcionalidades úteis que podem ser utilizadas em diversos programas
- Podemos acessar o i-ésimo argumento fornecido na linha de comando por `sys.argv[i]`
  - O tipo do argumento é sempre `str`
  - O nome do seu programa é armazendo em `sys.argv[0]`

Passando Argumentos na Linha de Comando

```
$ python meuprograma.py arg1 arg2 arg3
```

- Podemos passar quantos argumentos quisermos
- Para acessarmos os argumentos dentro do nosso programa, precisamos adicionar a linha `import sys`
  - O comando `import xxx` habilita o seu programa a usar funcionalidades implementadas na biblioteca `xxx`
  - Existem bibliotecas fornecidas pelos próprios criadores da linguagem Python (*biblioteca padrão*) e existem bibliotecas feitas por terceiros (empresas/hobbystas/você!)
  - Criamos bibliotecas com funcionalidades úteis que podem ser utilizadas em diversos programas
- Podemos acessar o i-ésimo argumento fornecido na linha de comando por `sys.argv[i]`
  - O tipo do argumento é sempre `str`
  - O nome do seu programa é armazendo em `sys.argv[0]`

Passando Argumentos na Linha de Comando

```
$ python meuprograma.py arg1 arg2 arg3
```

- Podemos passar quantos argumentos quisermos
- Para acessarmos os argumentos dentro do nosso programa, precisamos adicionar a linha `import sys`
  - O comando `import xxx` habilita o seu programa a usar funcionalidades implementadas na biblioteca `xxx`
  - Existem bibliotecas fornecidas pelos próprios criadores da linguagem Python (*biblioteca padrão*) e existem bibliotecas feitas por terceiros (empresas/hobbystas/você!)
  - Criamos bibliotecas com funcionalidades úteis que podem ser utilizadas em diversos programas
- Podemos acessar o i-ésimo argumento fornecido na linha de comando por `sys.argv[i]`
  - O tipo do argumento é sempre `str`
  - O nome do seu programa é armazendo em `sys.argv[0]`

```
import sys
arg1 = sys.argv[1]
arg2 = sys.argv[2]
arg3 = sys.argv[3]
```

Programa Stock

```
$ python stock_trade.py 35 ITUB3 89.90 100.34
```

```python
import sys
units = int(sys.argv[1])
ticker = sys.argv[2]
unit_cost = float(sys.argv[3])
unit_price = float(sys.argv[4])

cost = units * unit_cost
price = units * unit_price
spread = price - cost
perc = spread / cost * 100

print(f"{ticker}")
print("=" * 20)
print(f"Cost: {cost}")
print(f"Price: {price}")
print(f"Spread: {spread} ({perc:.2f} %)")
```

Stddraw

Bibliotecas extras

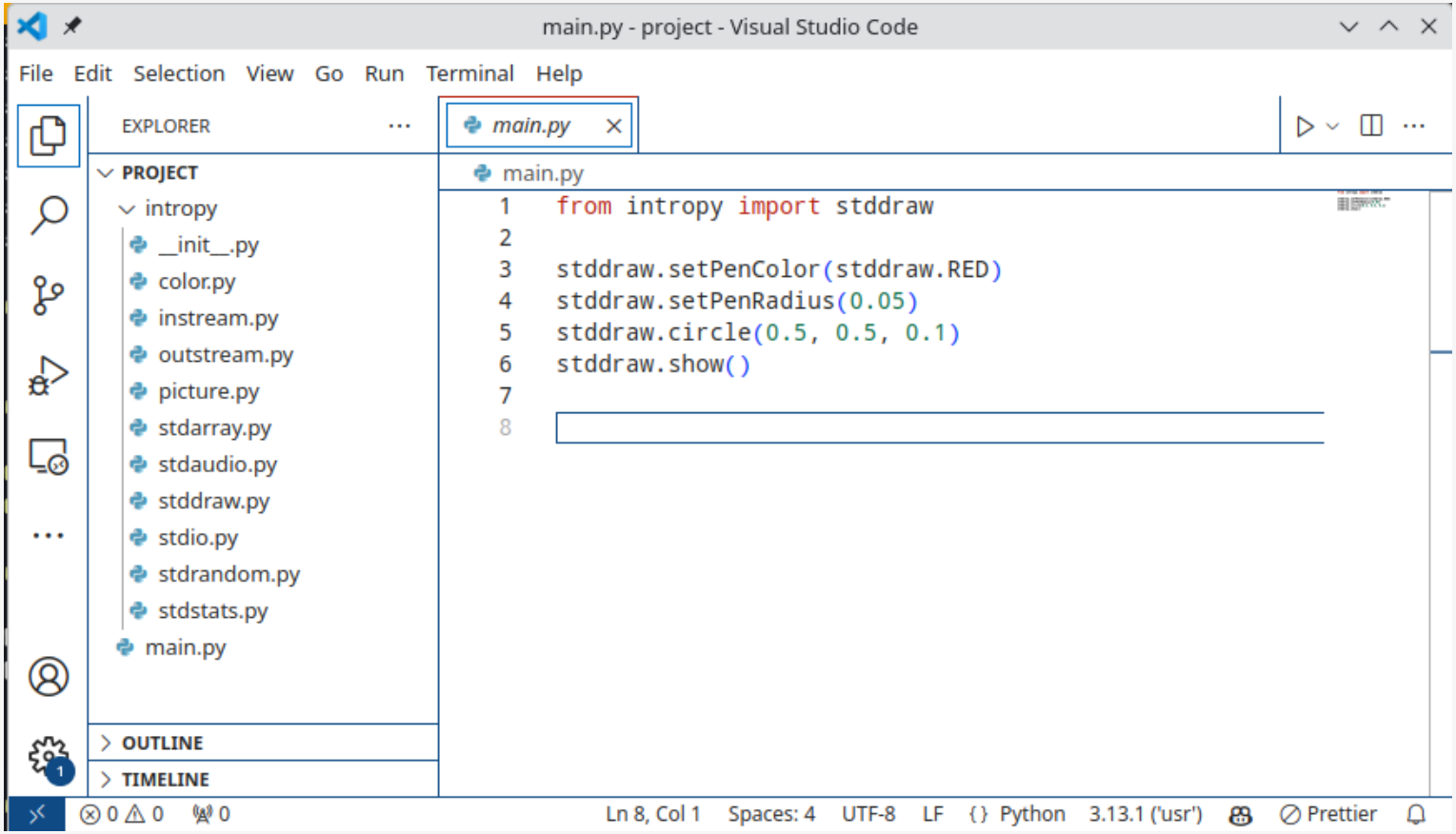
O livro *Introduction to programming in Python: an interdisciplinary approach* (2015), de Robert Sedgewick, Kevin Wayne, Robert Dondero introduz duas bibliotecas Python que iremos usar ao longo desse curso:

- `stddraw` contém funções para desenhar na tela
- `stdaudio` contém funções para gerar áudio

Essas bibliotecas não fazem parte da biblioteca padrão do Python, por isso precisaremos instalá-las

Instalando as Bibliotecas

- O arquivo `intropy.zip` no Moodle contém todos os arquivos da biblioteca
- Ao descompactar o arquivo, será criada uma pasta chamada `intropy`
- Para o seu código Python poder usar essa biblioteca, basta copiar o diretório `intropy` no mesmo diretório que contém o seu código



Módulo `stddraw`

O módulo `stddraw` é bem simples: permite desenhar linhas e pontos em uma tela 2D (*canvas*), que é exibida no final

Módulo `stddraw`

O módulo `stddraw` é bem simples: permite desenhar linhas e pontos em uma tela 2D (*canvas*), que é exibida no final

- Exibir o desenho em tempo real é muito custoso, por isso, programas que renderizam imagens, geralmente, criam a imagem na *background canvas* (uma tela que só existe na memória do computador) e, quando o desenho está pronto, requisitam que a *background canvas* seja renderizada na tela.

Módulo `stddraw`

Por padrão, enxergamos a *canvas* como o primeiro quadrante do sistema cartesiano, onde tanto as coordenadas de *x* quanto de *y* estão no intervalo $[0, 1]$.

- O ponto $(0, 0)$ encontra-se no canto inferior esquerdo
- O ponto $(1, 1)$ encontra-se no canto superior direito

Módulo `stddraw`

Por padrão, enxergamos a *canvas* como o primeiro quadrante do sistema cartesiano, onde tanto as coordenadas de *x* quanto de *y* estão no intervalo $[0, 1]$.

- O ponto $(0, 0)$ encontra-se no canto inferior esquerdo
- O ponto $(1, 1)$ encontra-se no canto superior direito

Função	Descrição
`stddraw.line(x0, y0, x1, y1)`	desenha uma linha de `(x0, y0)` até `(x1, y1)`
`stddraw.point(x, y)`	desenha um ponto em `(x, y)`
`stddraw.show()`	renderiza a *background canvas* na tela

Módulo `stddraw`

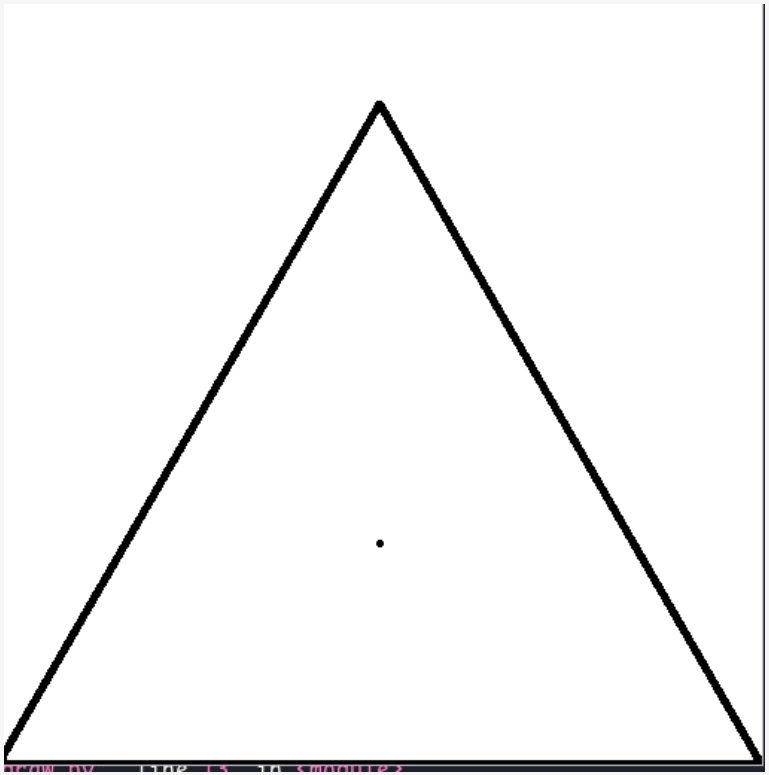
Por padrão, enxergamos a *canvas* como o primeiro quadrante do sistema cartesiano, onde tanto as coordenadas de x quanto de y estão no intervalo $[0, 1]$.

- O ponto $(0, 0)$ encontra-se no canto inferior esquerdo
- O ponto $(1, 1)$ encontra-se no canto superior direito

Função	Descrição
`stddraw.line(x0, y0, x1, y1)`	desenha uma linha de `(x0, y0)` até `(x1, y1)`
`stddraw.point(x, y)`	desenha um ponto em `(x, y)`
`stddraw.show()`	renderiza a *background canvas* na tela
`stddraw.setXscale(x0, x1)`	Define o intervalo de pontos no eixo x (default: [0, 1])
`stddraw.setYscale(y0, y1)`	Define o intervalo de pontos no eixo y (default: [0, 1])

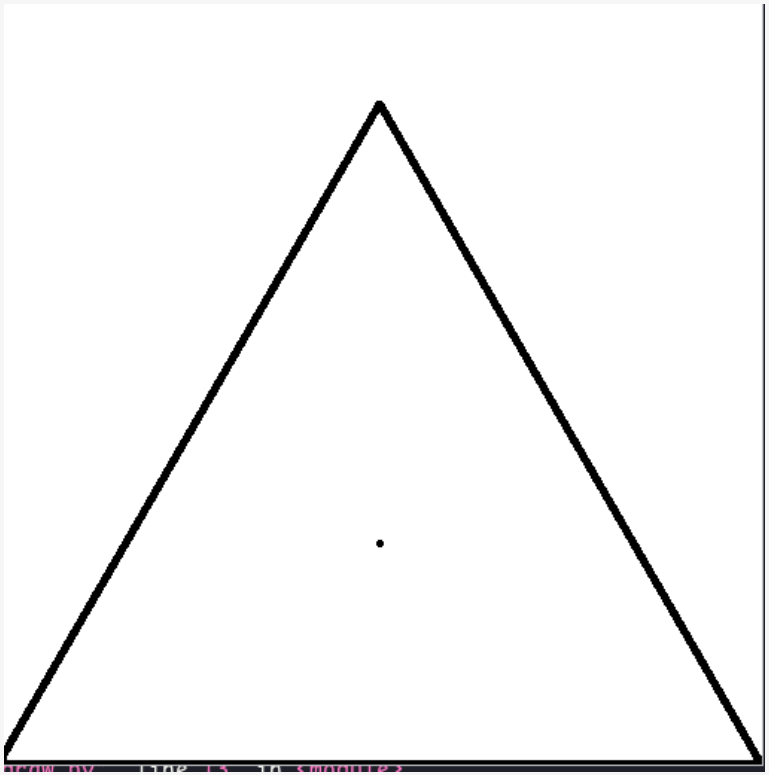
Hello World Gráfico

```
from intropy import stddraw

h = (3 ** 0.5) / 2
stddraw.line(0.0, 0.0, 1.0, 0.0)
stddraw.line(1.0, 0.0, 0.5, h)
stddraw.line(0.5, h, 0.0, 0.0)
stddraw.point(0.5, h/3.0)
stddraw.show()
```



Hello World Gráfico

```
from intropy import stddraw

h = (3 ** 0.5) / 2
stddraw.line(0.0, 0.0, 1.0, 0.0)
stddraw.line(1.0, 0.0, 0.5, h)
stddraw.line(0.5, h, 0.0, 0.0)
stddraw.point(0.5, h/3.0)
stddraw.show()
```



- Para habilitar o módulo `stddraw` usamos a instrução `from intropy import stddraw`

API do módulo `stddraw`

API (*Application Programming Interface*) é o termo genérico usado para estabelecer como um aplicativo/módulo/biblioteca pode se comunicar com outro. No caso de uma biblioteca, é a coleção de funções que ela provê.

API do módulo `stddraw`

API (*Application Programming Interface*) é o termo genérico usado para estabelecer como um aplicativo/módulo/biblioteca pode se comunicar com outro. No caso de uma biblioteca, é a coleção de funções que ela provê.

Função	Descrição
`stddraw.line(x0, y0, x1, y1)`	desenha uma linha de `(x0, y0)` até `(x1, y1)`
`stddraw.point(x, y)`	desenha um ponto em `(x, y)`
`stddraw.show()`	renderiza a *background canvas* na tela
`stddraw.setXscale(x0, x1)`	Define o intervalo de pontos no eixo *x* (default: [0, 1])
`stddraw.setYscale(y0, y1)`	Define o intervalo de pontos no eixo *y* (default: [0, 1])
`stddraw.setCanvasSize(w, h)`	Define o tamanho da *canvas* (default: w=h=512)
`stddraw.setPenRadius(r)`	Define o raio da caneta (default: r = 0.005)
`stddraw.setPenColor(r)`	Define a cor da caneta (default: BLACK)

Exercícios

Exercícios

1. Escreva uma calculadora de IMC, que é calculado da forma $IMC = \frac{p}{h^2}$, onde o p é o peso em quilos e h é a altura em metros.
2. Escreva um programa que simule o retorno de um investimento a juros compostos. O seu programa deve ler o montante inicial M, a taxa de juros mensal t e o número de meses n no qual o capital ficará imobilizado no investimento. Lembre-se que o montante final é computado como $MF = M \times (1 + t)^n$.
3. Escreva um programa chamado `seg2time.py`. O seu programa deve receber um número na linha de comando, que representa uma quantidade de segundos, e converter essa medida de tempo para dias, horas, minutos e segundos. Exemplo:

```
python set2time.py 18684708
dias: 216, horas: 6, minutos: 11, segundos: 48
```

4. Desenhe uma estrela estrela com 5 pontas