1.3 - Guia inicial de referências

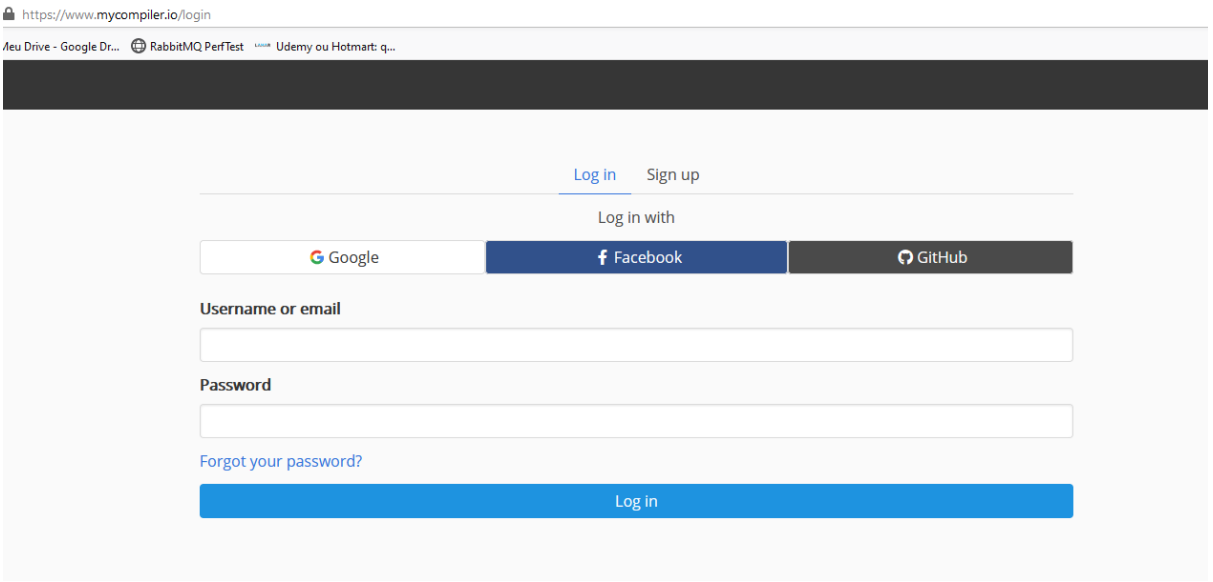
Site oficial da linguagem Python:

https://www.python.org/

Site do My Compiler:

https://www.mycompiler.io/

No MyCompiler você pode criar uma conta utilizando a sua conta Google ou Facebook. Ao criar uma conta, você pode salvar seus trabalhos:



Repositório de exemplos do curso:

https://github.com/cleuton/pythondrops/cursos/backend

Zen do Python:

Um conjunto de diretrizes que orientam o uso da linguagem:

Beautiful is better than ugly. (Bonito é melhor do que feio):

Escreva código simples e fácil de entender, em vez de código intrincado. Use variáveis explicativas, escreva seu código para que outros entendam.

Explicit is better than implicit. (Explícito é melhor que implícito):

Use nomes explícitos, precedidos pelos nomes de pacotes. Torne as conversões de dados explícitas. Evite soluções de indireção implícitas, como as utilizadas em Java, onde tudo é oculto por indireção, atrás de características obscuras ou de montes de frameworks. .

Simple is better than complex. (Simples é melhor que complexo) e Complex is better than complicated. (Complexo é melhor que complicado):

Devemos tentar resolver os problemas da maneira mais simples e óbvia possível, evitando o antipattern da “Complexidade acidental”, no qual a maior parte da complexidade tem origem na solução e não no problema.

Se uma solução simples não for possível, escolha uma solução que resolva o problema sem acrescentar muito mais complexidade do que a necessária.

Flat is better than nested.(Achatado - plano - é melhor do que aninhado):

Alguns programadores, especialmente os oriundos do mundo Java, adoram aninhar coisas. Criam categorias, subcategorias e sub-subcategorias de classes, na esperança de aumentar o reúso de seu código. Desta forma, criam coisas estranhas como 300 diretórios aninhados. Não é raro vermos pacotes java assim:

com.exemplo.sistema.modelo.abstrato.interfaces.negocio.frontend.Cliente

Evite aninhar funções e classes desnecessariamente. Prefira estruturas simples e lineares.

Sparse is better than dense. (Esparso é melhor do que denso):

A interpretação mais comum é quanto à dispersão das linhas de código. Alguns programadores adoram fazer soluções “one-liner”, onde tudo é feito em uma ou poucas linhas. Veja este exemplo:

import sys,os,re,fileinput;a=[i[2] for i in os.walk('.') if i[2]] [0];[sys.stdout.write(re.sub('at','op',j)) for j in fileinput.input(a,inplace=1)]

Bonito, impressionante, certo? Só que seus colegas lhe odiarão. Faça algo mais esparso, distribuindo melhor as linhas.

Readability counts. (Facilidade de leitura conta)

O que é um código fácil de ler? É aquele que dispensa comentários! Podemos facilmente acompanhar o que o programador quer fazer e trabalhar com o código. Temos vários elementos que facilitam a leitura, como:

- Nomes explicativos para funções e variáveis;
- Evitar encadear comandos em uma só linha;
- Evitar complexidade acidental;

Special cases aren't special enough to break the rules. (Casos especiais não são tão especiais para quebrar as regras)
Although practicality beats purity. (Embora a praticidade supere a pureza…):

Devemos seguir as regras e boas práticas de programação, desde que façam sentido no caso em que desejamos aplicá-las. Criar indireções e hierarquias de objetos desnecessariamente pode gerar Complexidade acidental, diminuindo a legibilidade do nosso código-fonte.

Use o bom senso para saber quando usar padrões e soluções, como: Indireções, Abstrações, Factories etc.

Errors should never pass silently. (Erros não devem jamais passarem silenciosos…)
Unless explicitly silenced. (A não ser que sejam explicitamente silenciados):

Aqui devemos pensar no princípio do “fail fast” (falhar rápido). Se algo deu errado, levante uma exceção e pronto! Nada de retornar “null” ou um código de erro obscuro. Silenciar exceções é a pior coisa que você pode fazer.

Se você quer ignorar determinados erros, capture-os, exiba mensagens e depois tome alguma atitude. Alguém sempre saberá que o erro aconteceu, mas você optou por ignorá-lo.

In the face of ambiguity, refuse the temptation to guess. (Em face à ambiguidade, rejeite a tentação de adivinhar):

Se algo saiu errado, use os métodos da Engenharia de código para verificar se seu código contém alguma falha. Recuse a tentação de imaginar outras causas e jogar a culpa em outros componentes.

É muito comum em alguns sistemas operacionais e plataformas, as pessoas saírem com soluções do tipo: Fecha a IDE e reboota o sistema que o erro some.

There should be one-- and preferably only one --obvious way to do it. (Deve haver uma - e preferencialmente só uma - maneira óbvia de fazer). Although that way may not be obvious at first unless you're Dutch. (Embora esse caminho possa não ser óbvio no início, a menos que você seja holandês):

Muitos programadores e linguagens oferecem mais de uma maneira de fazer a mesma coisa, e não é óbvio saber qual é a mais adequada ao seu caso. Em python procura-se adotar esta filosofia de prover uma única maneira óbvia de fazer algo.

A segunda parte é uma brincadeira de Guido Von Rossun, o inventor do python, que é Holandês, que, como inventor, tudo para ele é óbvio.

Now is better than never. (Agora é melhor do que nunca)
Although never is often better than *right* now. (Embora nunca seja frequentemente melhor que agora mesmo):

Estes ditados podem ser relacionados à procrastinação e à pressa. Por exemplo: “agora é melhor do que nunca” nos diz que devemos corrigir, escrever casos de teste e organizar o código em vez de procrastinar. E a frase: “Embora *nunca* seja frequentemente melhor que agora mesmo”, nos diz que devemos planejar e fazer as coisas com cuidado, evitando soluções apressadas.

Também podem estar relacionadas ao fato do seu código entrar em loop infinito ou retornar rapidamente um resultado incorreto.

If the implementation is hard to explain, it's a bad idea. (Se a implementação é difícil de explicar, é uma ideia ruim). If the implementation is easy to explain, it may be a good idea. (Se a implementação é fácil de explicar, pode ser uma boa ideia):

Se a implementação é difícil de explicar, então a solução está com alto índice de “gordura”, na forma de Complexidade Acidental, portanto, é uma ideia ruim, com certeza. Agora, se é fácil de explicar, pode ser uma boa ideia. Nem toda ideia fácil de explicar é boa, mas toda ideia difícil de explicar é ruim.

Namespaces are one honking great idea -- let's do more of those! (Os namespaces são uma ótima ideia - vamos fazer mais daqueles):

No ambiente diversificado de hoje, muitos programadores criam componentes que usamos em nosso código e podem acabar criando classes e scripts com o mesmo nome que usamos. Para evitar estes conflitos, o python adota o conceito de módulos que podemos importar e qualificar os nomes dentro do nosso código. Exemplo:

```
import utils.formatador

def formatador(p):
      return utils.formatador(p)
```

Você pode imprimir o “Zen do python” sempre que desejar, bastando digitar na janela interativa do python:

```
import this
```

Dicas iniciais sobre python

É uma linguagem interpretada. Para executar, você precisa ter o interpretador Python instalado. Veja aqui como instalar o python para vários sistemas operacionais:

https://www.python.org/downloads/

Para executar um programa basta:

```
python <programa>.py
```

Para entrar no python interativo:

```
python
```

Para sair do python interativo:

```
quit()
```

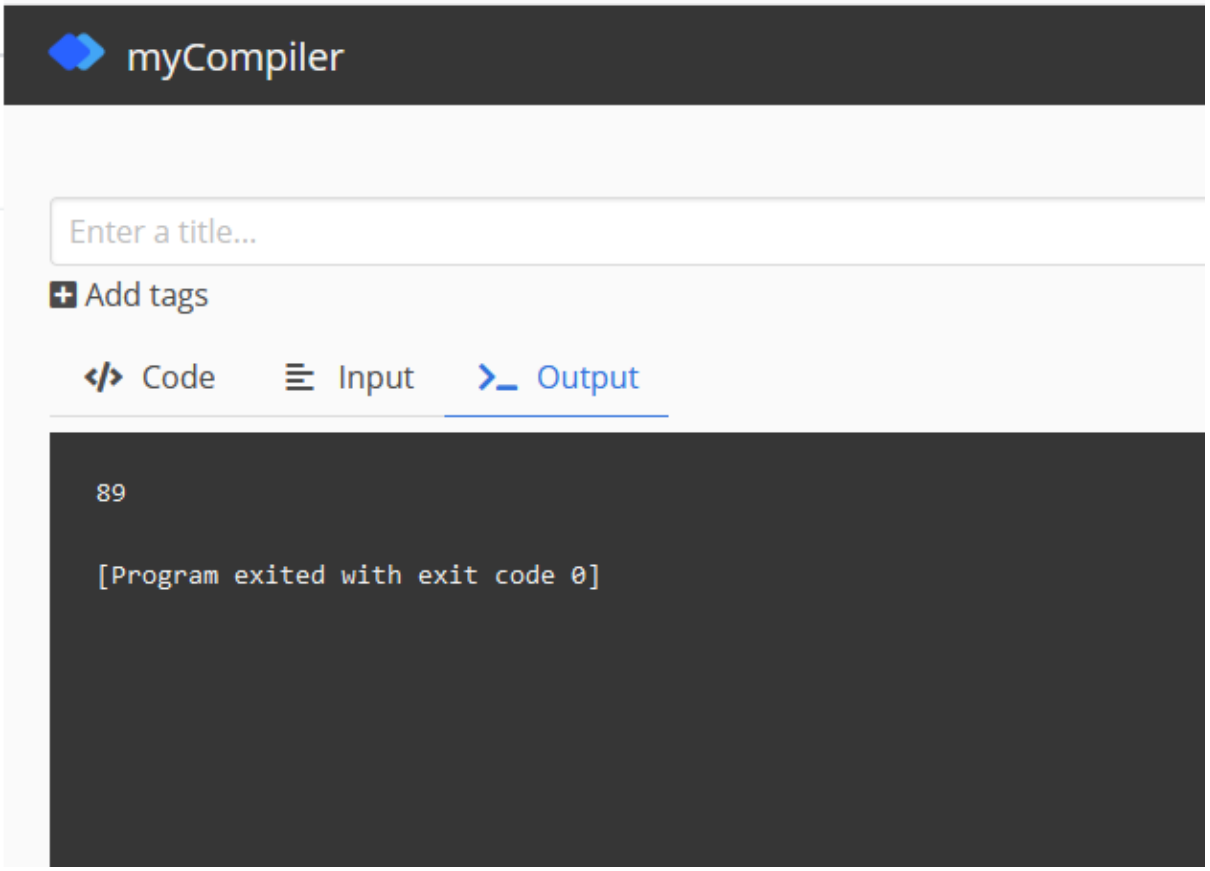
Um programa simples:

```
def execute(x):
      fibonacci = (lambda x, x_1=1, x_2=0:
      x_2 if x == 0
      else fibonacci(x - 1, x_1 + x_2, x_1))
      return fibonacci(x)

if __name__=='__main__':
      print(execute(11))
```

O resultado deve ser o número 89, que é o 11o termo da sequência de Fibonacci.

Vamos executar no MyCompiler. Acesse MyCompiler.io, selecione python, apague a linha que está no editor e digite o programa, exatamente como está listado acima. Depois clique em “Run”. O resultado deve ser este:



Entendendo o código

Declaramos uma função chamada “execute”, que recebe um parâmetro identificado como: “x”:

```
def execute(x):
```

Depois, invocamos esta função passando o argumento 11:

```
execute(11)
```

Na verdade, pedimos para mostrar na console o resultado da execução desta função:

```
print(execute(11))
```

Código imediato e código sob demanda

Em um programa python, todo código que não esteja dentro de declarações “def” é considerado código imediato e é executado imediatamente:

- Se o programa for executado;
- Se o programa for importado por outro programa;

Quando um programa é executado diretamente na console, a variável especial __name__ contém o valor “__main__”. Neste caso, queremos invocar a função.

Mas podemos querer incorporá-la em outro programa, através da diretiva “import” e, neste caso, não queremos executá-la de imediato.

Daí aquele “if” esquisito antes de invocar a função.

Expressões “lambda”

São funções para as quais passamos um código a ser executado. É um conceito de programação funcional. No nosso exemplo, criamos uma expressão lambda:

```
fibonacci = (lambda x, x_1=1, x_2=0:
     x_2 if x == 0
     else fibonacci(x - 1, x_1 + x_2, x_1))
```

A palavra-chave “lambda” indica que a seguir vem uma expressão lambda. Tudo antes do “:” são os argumentos passados para a função lambda, e tudo depois do “:” é a função lambda.

A função lambda poderia ser traduzida assim:

```
def fibonacci(x, x_1=1, x_2=0):
     if x == 0:
          return x_2
     else:
          fibonacci(x - 1, x_1 + x_2, x_1)
```

É só uma maneira de escrever funções. Escolha sempre a que for mais apropriada para sua necessidade.