No capítulo anterior fizemos os primeiros passos com o Python, desde a sua instalação e até vimos um pouco da sua sintaxe no console. Mas para escrever uma aplicação completa, utilizando o console, não parece ser uma boa ideia. Podemos ter um editor de texto que nos auxilie nessa programação, nos permitindo trabalhar com vários arquivos, auxiliando a nossa vida.

Há várias opções de editores de texto no mercado, entre elas o Atom e o Sublime Text. Apesar de esses editores nos ajudarem a escrever o código, eles não são focados no Python, e sim em dar suporte a várias linguagens. Então, vamos utilizar uma ferramenta (IDE, do inglês Integrated Development Environment) só focada para o Python, assim como existe o Eclipse para o Java.

Instalando o PyCharm

Uma IDE que é voltada exclusivamente para o Python é o PyCharm, e é ela que iremos utilizar aqui no treinamento. A sua instalação é bem simples, basta acessar a sessão de Download do site oficial, baixar e instalar a versão Community da IDE, já que a versão Professional é paga.

Conhecendo o PyCharm e criando o primeiro projeto

Após instalar o PyCharm, vamos abri-lo e criar o nosso primeiro projeto, clicando em Create New Project. Na tela que irá se abrir, nos é perguntado a localização do projeto e a versão do Python. Vamos criar o projeto jogos, dentro da pasta PycharmProjects mesmo, e nos atentar à versão do Python (caso você tenha mais de uma versão instalada) ela deve ser a versão 3.

O projeto ficará sendo exibido na esquerda, e para criar o primeiro arquivo Python dentro dele, basta clicar com o botão direito do mouse em cima dele e clicar em New -> Python File, vamos colocar o nome do arquivo de adivinhacao.py.

A fim de testes, vamos imprimir uma mensagem simples:

```
print("Bem vindo ao jogo de Adivinhação!")
```

Para executar o arquivo, clicamos em Run -> Run... no menu superior, e selecionamos o arquivo que acabamos de criar. O console do PyCharm é aberto e exibe a saída do nosso programa, que é a mensagem Bem vindo ao jogo de Adivinhação!.

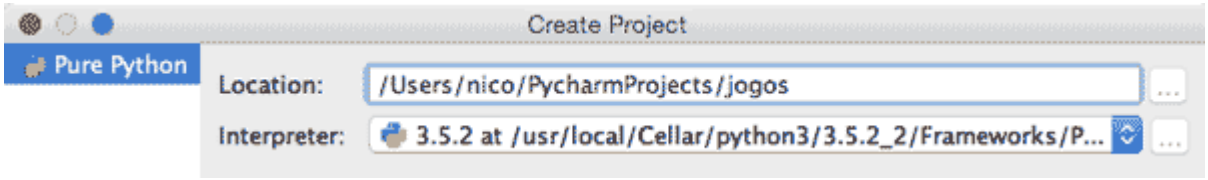
Em uma só tela, conseguimos ver os arquivos do projeto, o seu código fonte e o console, que exibe a saída do programa que for executado. Há vários outros recursos que ainda veremos mais à frente, mas o primeiro passo foi realizado!

Instalando o PyCharm

Baixe o PyCharm no site oficial. É importante escolher o seu sistema operacional e usar a versão Community da IDE. Após download instale o PyCharm no seu computador.

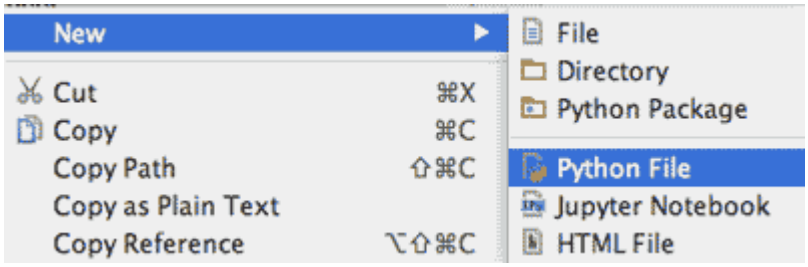
Criando o primeiro projeto

Abra o PyCharm e defina o diretório e o nome do projeto no campo Location. Para o nosso curso, vamos usar o nome jogos:

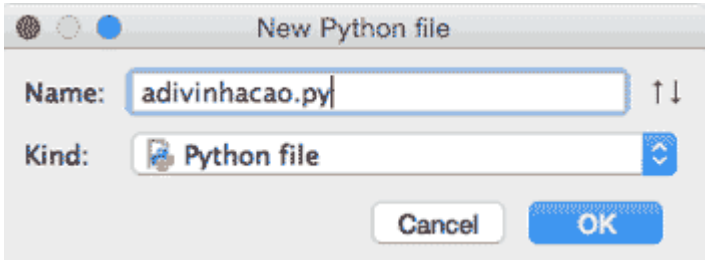


Primeiro arquivo

Agora crie o primeiro arquivo Python. Clique com o botão direito do mouse em cima do projeto e selecione New -> Python File:



Chame o arquivo de adivinhacao.py:

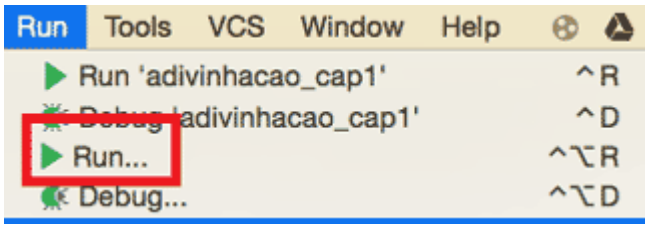


No arquivo, coloque uma mensagem simples:

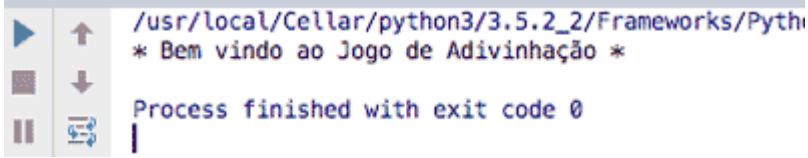
```
print("Bem vindo ao jogo de Adivinhação!")
```

Rodando Python

Agora rode o primeiro programa em Python, clicando em Run -> Run... no menu superior:



O console do PyCharm é aberto e exibe a saída do nosso programa:



A ideia do nosso jogo é termos que acertar um número secreto. Quando o programa estiver rodando, teremos que digitar um número e o programa dirá se acertamos ou erramos o número, com várias tentativas e níveis.

Vamos começar definindo esse número secreto (mais à frente vamos ver como gerar um número aleatório):

```
print("**********************************")
print("Bem vindo ao jogo de Adivinhação!")
print("**********************************")


numero_secreto = 42
```

Capturando a entrada do usuário

Agora, para que o usuário possa digitar o número, vamos utilizar a função input, ela trava o programa até que o usuário digite algo e tecle ENTER. Ela recebe por parâmetro a mensagem que será exibida no console e nos retorna o que o usuário digitou, logo vamos guardar esse resultado em uma variável, que chamaremos de chute:

```
print("**********************************")
print("Bem vindo ao jogo de Adivinhação!")
print("**********************************")


numero_secreto = 42


chute = input("Digite o seu número: ")
```

Para testar, vamos ao final do programa imprimir o conteúdo da variável chute, para mostrar realmente que o seu conteúdo será o que o usuário digitou:

```
print("**********************************")
print("Bem vindo ao jogo de Adivinhação!")
print("**********************************")


numero_secreto = 42


chute = input("Digite o seu número: ")
print("Você digitou: ", chute)
```

Podemos rodar o programa e ver que realmente é impresso o valor que digitarmos.

Comparando valores

Agora que conseguimos capturar o que o usuário digitou, precisamos comparar esse valor com o número secreto, para poder dizer ao usuário se ele digitou o número correto ou não. Bom, já sabemos o número secreto que o chute do usuário, então vamos comparar os dois, algo como:

```
se numero_secreto igual chute
    print("Você acertou!")
senão
    print("Você errou!")
```

Só que as palavras se, senão e igual não funcionam no mundo Python, temos que respeitar a sua sintaxe. O se em Python é `if`, o igual é a comparação `==` e o senão é `else`. Então, resumindo a sintaxe do Python é:

```
if (condição):
    executa código caso a condição seja verdadeira
else:
    executa código caso a condição seja falsa
```

Mas precisamos prestar atenção a alguns detalhes. É uma recomendação que a condição fique dentro de parênteses (apesar de também funcionar sem); para marcar o fim da instrução e início de um bloco (o código que será executado caso a condição seja verdadeira ou falsa), é utilizado dois pontos (:), e esse bloco obrigatoriamente deve estar 4 espaços (ou um TAB) mais à direita. Então o código ficará assim:

```
if (numero_secreto == chute):
    print("Você acertou!")
else:
    print("Você errou!")
```

Podemos rodar o programa e verificar que mesmo se digitarmos o número certo, recebemos a mensagem Você errou. Porque?

Convertendo uma string para número inteiro

Isso acontece porque a função `input` nos retorna uma string, pois qualquer coisa pode ser digitada, não é garantido que o usuário irá digitar um número. Como não há essa garantia, o retorno é uma string.

Já a variável `numero_secreto` é um número! Logo, do tipo inteiro. Então estamos testando a igualdade de um inteiro com uma string, logo essa comparação sempre será falsa, apesar da string representar um número inteiro. Para resolver isso precisamos mudar o tipo da variável, convertendo uma string em número inteiro.

Para isso, o Python possui a função `int`, que recebe um valor e o converte para inteiro, justamente o que queremos. Logo, vamos utilizá-la no nosso código:

```
print("**********************************")
print("Bem vindo ao jogo de Adivinhação!")
print("**********************************")
```

```
numero_secreto = 42

chute_str = input("Digite o seu número: ")
print("Você digitou: ", chute_str)
chute = int(chute_str)

if (numero_secreto == chute):
    print("Você acertou!")
else:
    print("Você errou!")
```

Agora a comparação é feita corretamente! Para sair do bloco do `else`, basta escrevermos algo depois dele, sem a indentação de 4 espaços:

```
print("**********************************")
print("Bem vindo ao jogo de Adivinhação!")
print("**********************************")

numero_secreto = 42

chute_str = input("Digite o seu número: ")
print("Você digitou: ", chute_str)
chute = int(chute_str)

if (numero_secreto == chute):
    print("Você acertou!")
else:
    print("Você errou!")

print("Fim do jogo")
```

Nesse vídeo veremos um pouco das diferenças entre o Python 3 e o Python 2, porque o Python 2 ainda é muito utilizado. Para esse vídeo, não é necessário instalar o Python 2, já que aqui só veremos as diferenças, a menos que você queira utilizá-lo.

A função print
A primeira diferença que podemos ver é que no Python 2, não precisamos colocar os parênteses na função print, eles são opcionais:

```
>>> print "python2"
python2
>>> print("python2")
python2
```

Já no Python 3, os parênteses são obrigatórios. Ainda na função print, no Python 2 não há os parâmetros sep e end, ao contrário do Python 3, e quando a função recebe mais de um valor, sua saída é diferente:

```
>>> print("python", "2")
('python', '2')
```

A função input
Outra diferença que podemos ver é na função input. Sabemos que no Python 3, essa função sempre retornará uma string. Já no Python 2, ela automaticamente converte o tipo da variável. Por exemplo:

```
>>> chute = input("Digite o seu número: ")
Digite o seu número: 42
>>> type(chute)
<type 'int'>
```

Isso foi considerado má prática, porque pode ou não ser a intenção do desenvolvedor converter o tipo da variável. Por isso é bem comum encontrar a função raw_input sendo utilizada no Python 2:

```
>>> chute = raw_input("Digite o seu número: ")
Digite o seu número: 42
>>> type(chute)
<type 'str'>
```

O retorno dessa função será sempre uma string, equivalente à função input do Python 3, mas ela não existe nessa versão.
Ao longo do treinamento veremos mais diferenças entre as versões!

No capítulo anterior começamos a implementar o jogo, vimos como capturar os dados digitados pelo usuário, como converter o valor e como fazer um if para saber se o usuário acertou ou não.

Nesse capítulo, vamos fazer com que o usuário possa dar vários chutes para tentar acertar o número, já que atualmente ele só tem uma tentativa. Mas antes disso, vamos implementar uma dica para o usuário, dizendo se o número que ele chutou é maior ou menor que o número secreto.

Para isso, precisamos mexer no bloco do else. Vamos ter que testar novamente, se o número for maior, imprimimos uma mensagem dizendo isso ao usuário, se for menos, diremos ao usuário que o número digitado é menor que o número secreto:

```
if (numero_secreto == chute):
    print("Você acertou!")
```

```
else:
    if (chute > numero_secreto):
        print("Você errou! O seu chute foi maior que o número secreto.")
    if (chute < numero_secreto):
        print("Você errou! O seu chute foi menor que o número secreto.")
```

Podemos testar e ver que tudo está funcionando perfeitamente.

else com condição de entrada

Podemos notar que, se o chute não for igual, nem maior que o número secreto, obviamente ele será menor, então o último `if` não é necessário:

```
if (numero_secreto == chute):
    print("Você acertou!")
else:
    if (chute > numero_secreto):
        print("Você errou! O seu chute foi maior que o número secreto.")
    else:
        print("Você errou! O seu chute foi menor que o número secreto.")
```

Mas para esses casos, podemos fazer um `else` com uma condição de entrada, o `elif`. Vamos utilizá-lo para deixar o código mais semântico, já que na prática não há diferença:

```
if (numero_secreto == chute):
    print("Você acertou!")
else:
    if (chute > numero_secreto):
        print("Você errou! O seu chute foi maior que o número secreto.")
    elif (chute < numero_secreto):
        print("Você errou! O seu chute foi menor que o número secreto.")
```

Melhorando a legibilidade do código

Podemos melhorar a legibilidade do nosso código, para que outros programadores que possam vir a desenvolver conosco o entendam melhor. Vamos deixar nossas condições mais claras, o que significa `chute == numero_secreto`, por exemplo? Que o usuário acertou, logo vamos extrair essa condição para uma variável:

```
acertou = chute == numero_secreto


if (acertou):
    print("Você acertou!")
```

```
else:
    if (chute > numero_secreto):
        print("Você errou! O seu chute foi maior que o número secreto.")
    elif (chute < numero_secreto):
        print("Você errou! O seu chute foi menor que o número secreto.")
```

Agora a condição if fica um pouco mais clara. Vamos fazer a mesma coisa para as outras duas condições:

```
acertou = chute == numero_secreto
maior = chute > numero_secreto
menor = chute < numero_secreto

if (acertou):
    print("Você acertou!")
else:
    if (maior):
        print("Você errou! O seu chute foi maior que o número secreto.")
    elif (menor):
        print("Você errou! O seu chute foi menor que o número secreto.")
```

Podemos testar e ver que tudo continua funcionando como antes, mas agora com um código um pouco mais legível. No próximo vídeo implementaremos a chance do usuário poder dar vários chutes para tentar acertar o número secreto. Até lá!

Queremos dar mais de uma oportunidade para o usuário tentar acertar o número secreto, já que é um jogo de adivinhação. A primeira ideia é repetir o código, desde a função input até o bloco do elif. Ou seja, para cada nova tentativa que quisermos dar ao usuário, copiaríamos esse código novamente.

Só que copiar código sempre é uma má prática, queremos escrever o nosso código apenas uma vez, e repeti-lo. Se queremos repetir o código, faremos um laço, ou um loop. O laço que queremos fazer é:

```
enquanto ainda há tentativas:
    chute_str = input("Digite o seu número: ")
    print("Você digitou: ", chute_str)
    chute = int(chute_str)

    acertou = numero_secreto == chute
    maior = chute > numero_secreto
    menor = chute < numero_secreto
```

```
    if (acertou):
        print("Você acertou!")
    else:
        if (maior):
            print("Você errou! O seu chute foi maior que o número secreto.")
        elif (menor):
            print("Você errou! O seu chute foi menor que o número secreto.")

print("Fim do jogo")
```

Só que o Python não entende português, então vamos traduzi-lo. A palavra tentativas será uma variável, chamaremos-a de total_de_tentativas:

```
total_de_tentativas = 3

enquanto ainda há total_de_tentativas:
    executa o código
```

A palavra enquanto no Python é o while, e assim como o if, ele recebe uma condição. A diferença é que o if, caso a condição seja verdadeira, executa apenas uma vez o código do seu bloco, já o while executa enquanto a condição for verdadeira:

```
total_de_tentativas = 3

while (ainda há total_de_tentativas):
    executa o código
```

Resta agora a expressão ainda há. A ideia é que o usuário tenha 3 tentativas, representada no código pela variável total_de_tentativas. A cada rodada subtraímos 1 do valor dessa variável, até o valor chegar a 0, que é quando devemos sair do while, logo vamos executá-lo enquanto a variável total_de_tentativas for maior que 0*:

```
total_de_tentativas = 3

while (total_de_tentativas > 0):
    chute_str = input("Digite o seu número: ")
    print("Você digitou: ", chute_str)
    chute = int(chute_str)

    acertou = numero_secreto == chute
    maior = chute > numero_secreto
    menor = chute < numero_secreto
```

```
    if (acertou):
        print("Você acertou!")
    else:
        if (maior):
            print("Você errou! O seu chute foi maior que o número secreto.")
        elif (menor):
            print("Você errou! O seu chute foi menor que o número secreto.")

print("Fim do jogo")
```

A condição está perfeita, falta, dentro do laço, subtrairmos 1 da variável total_de_tentativas:

```
total_de_tentativas = 3

while (total_de_tentativas > 0):
    chute_str = input("Digite o seu número: ")
    print("Você digitou: ", chute_str)
    chute = int(chute_str)

    acertou = numero_secreto == chute
    maior = chute > numero_secreto
    menor = chute < numero_secreto

    if (acertou):
        print("Você acertou!")
    else:
        if (maior):
            print("Você errou! O seu chute foi maior que o número secreto.")
        elif (menor):
            print("Você errou! O seu chute foi menor que o número secreto.")

    total_de_tentativas = total_de_tentativas - 1

print("Fim do jogo")
```

Testamos o código e ótimo, ele funciona! Mas pode ficar ainda melhor.

Representando a rodada

Vamos imprimir para o usuário qual o número da rodada que ele está jogando, para deixar claro quantas tentativas ele tem. Para isso vamos criar a variável `rodada`, que começa com o valor 1:

```
total_de_tentativas = 3
rodada = 1
```

E vamos imprimi-la antes do usuário digitar o seu chute:

```
total_de_tentativas = 3
rodada = 1

while (total_de_tentativas > 0):
    print("Tentativa", rodada, "de", total_de_tentativas)
    chute_str = input("Digite o seu número: ")

    # restante do código comentado
```

E para a variável `total_de_tentativas` continuar com o valor 3, não vamos mais subtrair 1 do seu valor, e sim adicionar 1 ao valor da variável `rodada`:

```
total_de_tentativas = 3
rodada = 1

while (total_de_tentativas > 0):
    print("Tentativa", rodada, "de", total_de_tentativas)
    chute_str = input("Digite o seu número: ")
    print("Você digitou: ", chute_str)
    chute = int(chute_str)

    acertou = numero_secreto == chute
    maior = chute > numero_secreto
    menor = chute < numero_secreto

    if (acertou):
        print("Você acertou!")
    else:
        if (maior):
            print("Você errou! O seu chute foi maior que o número secreto.")
        elif (menor):
            print("Você errou! O seu chute foi menor que o número secreto.")
```

```
    rodada = rodada + 1


print("Fim do jogo")
```

Por fim, precisamos modificar a condição, como o `total_de_tentativas` permanecerá com o valor 3, o código precisa ficar executando enquanto o valor da rodada for menor ou igual ao total de tentativas:

```
total_de_tentativas = 3
rodada = 1


while (rodada <= total_de_tentativas):
    print("Tentativa", rodada, "de", total_de_tentativas)
    chute_str = input("Digite o seu número: ")


    # restante do código comentado
```

Agora conseguimos imprimir para o usuário quantas tentativas restantes ele possui!

Com a lógica de tentativas implementada, vamos focar na impressão do número de tentativas para o usuário. Atualmente ela está assim:

```
print("Tentativa", rodada, "de", total_de_tentativas)
```

Desse jeito a frase é impressa do jeito que queremos, mas tem uma forma mais elegante de imprimir essa frase. Podemos deixar a string toda no código, dizendo onde que ela eventualmente pode mudar, no nosso caso é nos números. Onde a string pode mudar, colocamos chaves (`{}`):

```
print("Tentativa {} de {}")
```

As chaves significam que o Python deve substituí-las pelos valores das variáveis, então vamos passá-las:

```
print("Tentativa {} de {}", rodada, total_de_tentativas)
```

Se executarmos o programa, a seguinte frase é impressa:

```
Tentativa {} de {} 1 3
```

Não é exatamente isso que queremos, as primeiras chaves devem receber o valor da rodada, e as segundas o total de tentativas. Para isso funcionar, devemos chamar uma função baseada nessa string, a função `format`, passando para ela as variáveis que devem ficar no lugar das chaves:

```
print("Tentativa {} de {}".format(rodada, total_de_tentativas))
```

Podemos testar e ver que agora está tudo funcionando como antes! O que acabamos de fazer se chama interpolação de strings, muito comum nas linguagens e que nos oferece recursos da string para fazermos essas substituições.
Assim o nosso código fica um pouco mais elegante, já que nele vemos a string inteira, sabendo exatamente onde ela será alterada.

Voltando ao código do nosso jogo de adivinhação, implementamos o loop while, no qual temos uma variável rodada que começa com o valor 1, e é incrementada dentro do loop, que por sua vez tem uma condição de entrada, que é a rodada ser menor ou igual ao total de tentativas, que é 3.

Ou seja, a rodada tem um valor inicial, que é 1, e vai até 3. Fazemos um laço começando com um valor inicial, até um valor final, sempre incrementando esse valor a cada iteração.
Em casos como esse, existe um outro loop que simplifica essa ideia de começar com um valor, e incrementá-lo até chegar em um valor final, o loop for.

Entendendo o for

Para entender o loop for, podemos ir até o console do Python para ver o seu funcionamento. A ideia é nós definirmos o valor inicial e o valor final, que o loop o incrementa automaticamente. Para definir o valor inicial e final, utilizamos a função range, passando-os por parâmetro, definindo assim a série de valores. A sintaxe é a seguinte

```
>>> para variável em série de valores:
...     faça algo
```

Isso, em Python, pode ficar assim:

```
>>> for rodada in range(1,10):
...
```

Na primeira iteração, o valor da variável rodada será 1, depois 2 e até chegar ao valor final da função range menos 1, isto é, o segundo parâmetro da função não é inclusivo. No exemplo acima, a série de valores é de 1 a 9. Podemos confirmar isso imprimindo o valor da variável rodada:

```
>>> for rodada in range(1,10):
...     print(rodada)
...
1
2
3
4
5
6
7
8
9
```

Com a função range, podemos definir um step, que é o intervalo entre os elementos, por padrão o step é 1. Definimos-o passando um terceiro parâmetro para a função:

```
>>> for rodada in range(1,10,2):
...     print(rodada)
...
```

```
1
3
5
7
9
```

Mas não necessariamente precisamos usar a função `range` no `for`, podemos passar os valor manualmente:

```
>>> for rodada in [1,2,3,4,5]:
...     print(rodada)
...
1
2
3
4
5
```

O resultado é o mesmo, mas o código fica mais verboso.

Utilizando o for no jogo

Voltando ao nosso jogo, não vamos mais utilizar o `while`, e sim o `for`, começando no 1 e indo até o total de tentativas. Para isso precisamos remover a declaração da variável `rodada*` e o seu incremento dentro do loop:

```
numero_secreto = 42
total_de_tentativas = 3


for rodada in range(1, total_de_tentativas):
    print("Tentativa {} de {}".format(rodada, total_de_tentativas))
    chute_str = input("Digite o seu número: ")
    print("Você digitou: ", chute_str)
    chute = int(chute_str)

    acertou = numero_secreto == chute
    maior = chute > numero_secreto
    menor = chute < numero_secreto

    if (acertou):
        print("Você acertou!")
```

```
    else:
        if (maior):
            print("Você errou! O seu chute foi maior que o número secreto.")
        elif (menor):
            print("Você errou! O seu chute foi menor que o número secreto.")

print("Fim do jogo")
```

É importante saber que o `for` não deve ter parênteses.
Podemos testar e ver que só fizemos 2 tentativas. Isso porque, como foi falado anteriormente, o segundo parâmetro da função `range` não é inclusivo, no caso do nosso jogo, `range(1,3) irá gerar a série 1 e 2 somente. Logo vamos somar 1 ao total de tentativas dentro da função `range`:

```
numero_secreto = 42
total_de_tentativas = 3

for rodada in range(1, total_de_tentativas + 1):
    print("Tentativa {} de {}".format(rodada, total_de_tentativas))
    chute_str = input("Digite o seu número: ")
    print("Você digitou: ", chute_str)
    chute = int(chute_str)

    acertou = numero_secreto == chute
    maior = chute > numero_secreto
    menor = chute < numero_secreto

    if (acertou):
        print("Você acertou!")
    else:
        if (maior):
            print("Você errou! O seu chute foi maior que o número secreto.")
        elif (menor):
            print("Você errou! O seu chute foi menor que o número secreto.")

print("Fim do jogo")
```

Agora podemos testar novamente o nosso jogo, e ver que tudo está funcionando perfeitamente!

No nosso jogo, sabemos que o número secreto é fixo e definido com o valor 42, por enquanto. Vamos jogar e digitar esse valor de primeira:

```
**********************************
Bem vindo ao jogo de Adivinhação!
**********************************
Tentativa 1 de 3
Digite o seu número: 42
Você digitou:  42
Você acertou!
Tentativa 2 de 3
Digite o seu número:
```

Acertamos o número, mas ainda temos uma segunda e terceira tentativas! Não faz muito sentido isso né? Se nós ganhamos, temos que parar as rodadas, não devemos continuar.

Parando o laço

Dentro do if, se acertarmos, devemos parar e sair do laço. Para isso existe um comando do Python, assim como outras linguagens, o break, que faz com que saiamos do laço:

```
if (acertou):
    print("Você acertou!")
    break
else:
    if (maior):
        print("Você errou! O seu chute foi maior que o número secreto.")
    elif (menor):
        print("Você errou! O seu chute foi menor que o número secreto.")
```

Podemos agora jogar novamente e...:

```
**********************************
Bem vindo ao jogo de Adivinhação!
**********************************
Tentativa 1 de 3
Digite o seu número: 42
Você digitou:  42
Você acertou!
Fim do jogo
```

Ótimo! Acertamos o número e o jogo foi encerrado, sem mais rodadas.

Limitando o número a ser digitado

Vamos limitar o número que o usuário deve digitar, de 1 a 100. Vamos deixar isso claro para ele alterando a mensagem do input:

```
for rodada in range(1, total_de_tentativas + 1):
    print("Tentativa {} de {}".format(rodada, total_de_tentativas))
    chute_str = input("Digite um número entre 1 e 100: ")
    print("Você digitou: ", chute_str)
    chute = int(chute_str)

    ## resto do código comentado
```

Só que agora não devemos aceitar valores fora desse limite, logo vamos verificar o número digitado, e se ele for menor que 1 OU (em Python, a palavra chave or) maior que 100, vamos exibir uma mensagem para o usuário:

```
for rodada in range(1, total_de_tentativas + 1):
    print("Tentativa {} de {}".format(rodada, total_de_tentativas))
    chute_str = input("Digite um número entre 1 e 100: ")
    print("Você digitou: ", chute_str)
    chute = int(chute_str)

    if (chute < 1 or chute > 100):
        print("Você deve digitar um número entre 1 e 100!")

## resto do código comentado
```

Mas não faz sentido continuarmos executando o código do loop se o valor não estiver no intervalo exigido. O que queremos não é sair do laço, e sim continuar para a próxima rodada, acabando com a iteração. Para isso existe a palavra chave continue:

```
for rodada in range(1, total_de_tentativas + 1):
    print("Tentativa {} de {}".format(rodada, total_de_tentativas))
    chute_str = input("Digite um número entre 1 e 100: ")
    print("Você digitou: ", chute_str)
    chute = int(chute_str)

    if (chute < 1 or chute > 100):
        print("Você deve digitar um número entre 1 e 100!")
        continue

## resto do código comentado
```

Esse comando faz com que a iteração do laço acabe, e comece a próxima. Vamos testar:

```
**********************************
Bem vindo ao jogo de Adivinhação!
**********************************
Tentativa 1 de 3
Digite um número entre 1 e 100: 0
Você digitou:  0
Você deve digitar um número entre 1 e 100!
Tentativa 2 de 3
Digite um número entre 1 e 100:
```

Perfeito! O número digitado era incorreto, então fomos para a próxima tentativa.

Então vimos aqui o `break`, que acaba, encerra o laço; e o `continue`, que acaba, encerra a iteração, continuando para a próxima.

A lógica principal do nosso jogo já está funcionando, mas ainda há um detalhe, o número secreto não é tão secreto assim, pois ele está fixo! Então vamos alterar isso, para que ele passe a ser um número aleatório, coisa que veremos nesse capítulo.

Gerando um número aleatório

A ideia é que o próprio jogo, toda vez que for executado, gere esse número, ele que decide isso, não nós. E para gerar um número aleatório, o Python 3 possui a função `random()`, que gera um número no intervalo entre 0.0 e 1.0. Mas ao contrário das funções built-in do Python, como as funções `input()`, `int()`, `print()` e `range()`, que são funções embutidas do Python (que já vem com o mesmo), a função `random` não vem, pois está em um módulo separado, e esse módulo precisa ser importado.

Podemos ir ao console do Python e testar isso. Primeiro importando o módulo:

```
>>> import random
```

E a partir desse módulo, chamamos a função `random()`:

```
>>> import random
>>> random.random()
0.6022965518496559
```

Arredondando um número

Só que, como podemos perceber, o número gerado tem muitas casas decimais e está no intervalo entre 0.0 e 1.0, mas no nosso jogo precisamos de um número entre 1 e 100. O que podemos fazer é multiplicar o número gerado por 100:

```
>>> import random
```

```
>>> random.random() * 100
58.30742817094118
```

Já conseguimos chegar a um número mais próximo do ideal, falta agora removermos as casas decimais. Podemos utilizar a já conhecida função int, que irá converter o número aleatório, que é um float, em um número inteiro:

```
>>> int(random.random() * 100)
91
```

Mas reparem no exemplo abaixo:

```
>>> numero_random = random.random() * 100
>>> numero_random
18.895629671768187
>>> int(numero_random)
18
```

A função int nada mais faz do que remover as casas decimais do número flutuante. Mas o número gerado acima está mais próximo de 19 do que de 18, correto? Será que temos uma função que arredonda esse número para nós? Sim! Temos mais uma função built-in, a round:

```
>>> numero_random = random.random() * 100
>>> numero_random
18.895629671768187
>>> int(numero_random)
18
>>> round(numero_random)
19
```

Conhecendo isso, podemos aplicar ao nosso jogo. Faremos isso no próximo vídeo, até lá!

Para gerar um número aleatório no nosso jogo, a primeira coisa que devemos fazer é importar o seu módulo, no início do programa:

```
import random

print("**********************************")
print("Bem vindo ao jogo de Adivinhação!")
print("**********************************")

numero_secreto = 42
total_de_tentativas = 3

# restante do código comentado
```

Com o módulo importado, vamos remover o valor fixo da variável numero_secreto e substituir por um valor aleatório que será gerado pela função random():

```
import random


print("**********************************")
print("Bem vindo ao jogo de Adivinhação!")
print("**********************************")


numero_secreto = random.random()
total_de_tentativas = 3


# restante do código comentado
```

Mas não podemos nos esquecer de multiplicar esse número por 100 e arredondá-lo:

```
import random


print("**********************************")
print("Bem vindo ao jogo de Adivinhação!")
print("**********************************")


numero_secreto = round(random.random() * 100)
total_de_tentativas = 3


# restante do código comentado
```

Perfeito, conseguimos aplicar a mudança ao nosso código. Agora fica mais difícil de acertar o número secreto, até para nós, os desenvolvedores do jogo :)

Gerando um número aleatório dentro de um intervalo

A ideia de multiplicar o número por 100 parece funcionar, mas podemos lembrar que o número gerado é entre 0.0 e 1.0, que quando multiplicado por 100 fica entre 0 e 100. Só que o nosso jogo não aceita o 0!

O ideal seria que pudéssemos definir um intervalo, dizer que queremos que o número gerado esteja entre 1 e 100. Como o random é um módulo, ele possui mais de uma função e a função randrange() serve exatamente para esse nosso problema. Se passarmos um parâmetro para ela, ela irá gerar um número inteiro de 0 até o valor desse parâmetro menos 1. Se passarmos dois parâmetros para ela, ela irá gerar um número inteiro do valor do primeiro parâmetro até o valor do segundo parâmetro menos 1, exatamente o que queremos!

Vamos, passando o intervalo que queremos para a função randrange(), lembrando que como queremos que o número gerado esteja entre 1 e 100 (inclusive), precisamos passar o número 101 como segundo parâmetro para a função:

```
import random


print("**********************************")
print("Bem vindo ao jogo de Adivinhação!")
print("**********************************")


numero_secreto = random.randrange(1, 101)
total_de_tentativas = 3


# restante do código comentado
```

Ótimo, problema resolvido!

No próximo capítulo vamos focar na dificuldade do jogo. Um jogo mais fácil terá um número de tentativas maior, e um mais difícil terá um número de tentativas menor. Até lá!

Neste capítulo, vamos fazer com que o nosso número secreto seja gerado de modo aleatório, ao invés de ser o número 42 fixo, que era até agora.

Importando a biblioteca random

Como vimos, se queremos ter acesso às funções de geração de números aleatórios, precisamos utilizar as funções da biblioteca random do Python. Então nosso primeiro passo é importá-la logo no começo do nosso arquivo:

```
import random


print("**********************")
...
## Resto do código
```

Gerando um número aleatório

Agora que temos acesso às funções de número aleatório, podemos pedir um número que atenda ao nosso pré-requisito, que é ser um número que está no intervalo de 1 até 100. Para isso, já conhecemos diversos modos, como por exemplo utilizar a função random.random() e depois multiplicar o valor por 100.

Mas nós também vimos a função random.randrange(), que é uma função que facilita a nossa vida, pois ela aceita como primeiro parâmetro o menor número que queremos gerar, no nosso caso o número 1 e como segundo parâmetro até qual número queremos que o nosso intervalo de números

gerados alcance, sem incluí-lo. Como queremos que o maior número seja o 100, vamos substituir a atribuição da nossa variável numero_secreto pela chamada da função, deste modo:

```
numero_secreto = random.randrange(1,101)
```

Testando a aleatoriedade

Podemos colocar um pequeno print abaixo, apenas para testar se o nosso número está sendo calculado mesmo:

```
numero_secreto = random.randrange(1,101)
print(numero_secreto)
```

Faça alguns testes e verifique se está tudo correto! Não esqueça depois de remover o print de teste, se não nosso usuário irá conseguir descobrir o número muito facilmente!

Vamos adicionar níveis ao nosso jogo, e conforme o nível vai ficando mais difícil, menos tentativas o usuário terá.

Começaremos perguntando ao usuário qual nível de dificuldade ele quer:

```
import random

print("**********************************")
print("Bem vindo ao jogo de Adivinhação!")
print("**********************************")

numero_secreto = random.randrange(1, 101)
total_de_tentativas = 3

print("Qual o nível de dificuldade?")
print("(1) Fácil (2) Médio (3) Difícil")

# resto do código comentado
```

E capturaremos o que ele digitar, já convertendo o valor para inteiro e guardando em uma variável:

```
import random

print("**********************************")
print("Bem vindo ao jogo de Adivinhação!")
print("**********************************")
```

```
numero_secreto = random.randrange(1, 101)
total_de_tentativas = 3

print("Qual o nível de dificuldade?")
print("(1) Fácil (2) Médio (3) Difícil")

nivel = int(input("Defina o nível: "))

# resto do código comentado
```

Agora falta mudar o total de tentativas baseado no nível que o usuário escolher. A variável será inicializada com 0, e faremos um if para verificar o nível escolhido, se o ele for fácil, o usuário terá 20 tentativas, se for médio terá 10, e se for difícil terá 5 tentativas:

```
import random

print("**********************************")
print("Bem vindo ao jogo de Adivinhação!")
print("**********************************")

numero_secreto = random.randrange(1, 101)
total_de_tentativas = 0

print("Qual o nível de dificuldade?")
print("(1) Fácil (2) Médio (3) Difícil")

nivel = int(input("Defina o nível: "))

if (nivel == 1):
    total_de_tentativas = 20
elif (nivel == 2):
    total_de_tentativas = 10
else:
    total_de_tentativas = 5

# resto do código comentado
```

Com isso conseguimos definir os níveis de dificuldade no nosso jogo. No próximo vídeos definiremos pontuação!

Com os níveis definidos, vamos agora calcular uma pontuação. Ela funcionará da seguinte maneira, o usuário começa o jogo com 1000 pontos, e a cada rodada que ele não acerta o número secreto, ele perderá pontos. Quanto mais distante for o chute, mais pontos o usuário irá perder. Por exemplo, se o número secreto for 40, e o usuário chutar 20, ele irá perder 20 pontos, que corresponde à distância entre os valores.

Vamos começar definindo a variável com 1000 pontos:

```
pontos = 1000
```

Após isso, o usuário irá perder pontos caso ele erre o chute, logo temos que implementar isso dentro da condição `else`:

```
if (acertou):
    print("Você acertou!")
    break
else:
    if (maior):
        print("Você errou! O seu chute foi maior que o número secreto.")
    elif (menor):
        print("Você errou! O seu chute foi menor que o número secreto.")
```

Vamos definir a variável `pontos_perdidos`, que subtrai o chute do número secreto:

```
if (acertou):
    print("Você acertou!")
    break
else:
    if (maior):
        print("Você errou! O seu chute foi maior que o número secreto.")
    elif (menor):
        print("Você errou! O seu chute foi menor que o número secreto.")
    pontos_perdidos = numero_secreto - chute
```

Depois, vamos subtrair os pontos perdidos da pontuação total:

```
if (acertou):
    print("Você acertou!")
    break
else:
    if (maior):
        print("Você errou! O seu chute foi maior que o número secreto.")
    elif (menor):
        print("Você errou! O seu chute foi menor que o número secreto.")
```

```
    pontos_perdidos = numero_secreto - chute
    pontos = pontos - pontos_perdidos
```

Isso funciona caso o usuário chute um número menor que o número secreto, mas e se for maior? Por exemplo, se o número secreto for 40 e o usuário chutar 60, de acordo com o cálculo do nosso código os pontos perdidos serão -20, e ao subtrair esse valor da pontuação total, ela irá aumentar! Então queremos fazer a subtração dos pontos perdidos, mas caso essa subtração tenha como resultado um número negativo, queremos que "esquecer" esse sinal, queremos sempre o número absoluto.
E para extrair o número absoluto, existe mais uma função built-in, a `abs()`:

```
if (acertou):
    print("Você acertou!")
    break
else:
    if (maior):
        print("Você errou! O seu chute foi maior que o número secreto.")
    elif (menor):
        print("Você errou! O seu chute foi menor que o número secreto.")
    pontos_perdidos = abs(numero_secreto - chute)
    pontos = pontos - pontos_perdidos
```

Por fim, falta exibirmos a pontuação final ao usuário. Vamos alterar a mensagem de acerto do usuário, acrescentando a pontuação total. Faremos uso novamente da interpolação de strings:

```
if (acertou):
    print("Você acertou e fez {} pontos!".format(pontos))
    break
else:
    if (maior):
        print("Você errou! O seu chute foi maior que o número secreto.")
    elif (menor):
        print("Você errou! O seu chute foi menor que o número secreto.")
    pontos_perdidos = abs(numero_secreto - chute)
    pontos = pontos - pontos_perdidos
```

Com isso chegamos ao final do nosso jogo! No próximo capítulo veremos um pouco sobre linguagens compiladas e interpretadas, entre outros assuntos. Até lá!

Se conseguimos executar o jogo dentro do PyCharm, também conseguimos rodar o jogo na linha de comando, no terminal. Dentro do diretório do projeto jogos, basta executar:

```
python3 adivinhacao.py
```

No próximo treinamento, criaremos mais um jogo, a forca. Então já vamos deixar o seu arquivo preparado, criando o forca.py, também dentro do projeto jogos. Dentro desse arquivo, vamos deixar as mensagens de início e fim de jogo, semelhante ao jogo de adivinhação:

```
print("**********************************")
print("***Bem vindo ao jogo da Forca!***")
print("**********************************")


print("Fim do jogo")
```

Oferecendo todos os jogos ao usuário

Vamos oferecer os dois jogos ao usuário, ou seja, devemos perguntar ao usuário qual jogo ele quer executar, jogar. Mas onde vamos colocar essa funcionalidade? A ideia é não misturar os jogos, deixar cada um em um arquivo separado. Então vamos criar um novo arquivo com essa funcionalidade, o arquivo jogos.py, perguntando qual jogo ele quer escolher jogar:

```
print("**********************************")
print("*******Escolha o seu jogo!*******")
print("**********************************")


print("(1) Forca (2) Adivinhação")
```

Agora vamos capturar a opção do usuário e verificar qual jogo ele escolheu:

```
print("**********************************")
print("*******Escolha o seu jogo!*******")
print("**********************************")


print("(1) Forca (2) Adivinhação")


jogo = int(input("Qual jogo? "))


if (jogo == 1):
   print("Jogando forca")
elif (jogo == 2):
   print("Jogando adivinhação")
```

Importando arquivos

Ótimo, mas se queremos chamar um arquivo dentro de outro, precisamos importá-lo, algo parecido com o que fizemos com o módulo `random`:

```
import forca
import adivinhacao

print("**********************************")
print("*******Escolha o seu jogo!*******")
print("**********************************")

print("(1) Forca (2) Adivinhação")

jogo = int(input("Qual jogo? "))

if (jogo == 1):
   print("Jogando forca")
elif (jogo == 2):
   print("Jogando adivinhação")
```

O problema do import
Podemos executar para ver como está ficando:

```
**********************************
***Bem vindo ao jogo da Forca!***
**********************************
Fim do jogo
**********************************
Bem vindo ao jogo de Adivinhação!
**********************************
Qual o nível de dificuldade?
(1) Fácil (2) Médio (3) Difícil
Defina o nível:
```

Ué, o que aconteceu? Quando importamos um arquivo no Python, ele o executa! Podemos reparar que ele executou o arquivo forca.py e logo depois o adivinhacao.py. Mas obviamente não queremos isso, só queremos executar o arquivo quando nós quisermos! E é isso que faremos no próximo vídeo.

No vídeo anterior, vimos o problema do `import` acontecendo. Para resolver isso, vamos fazer algo semelhante à função `print()`, ela existe, mas só é executada quando chamada. Ou seja, vamos criar uma função específica para cada jogo.

Colocando o código dos jogos dentro de funções

Em Python, quando queremos criar uma função, precisamos defini-la, através da palavra chave def. Vamos começar definindo a função jogar() no arquivo forca.py:

```
# forca.py


def jogar():


    print("**********************************")
    print("***Bem vindo ao jogo da Forca!***")
    print("**********************************")


    print("Fim do jogo")
```

Faremos a mesma coisa no jogo de adivinhação, colocando todo o seu código dentro da função específica jogar():

```
# adivinhacao.py


import random


def jogar():
    print("**********************************")
    print("Bem vindo ao jogo de Adivinhação!")
    print("**********************************")


    numero_secreto = random.randrange(1, 100)
    total_de_tentativas = 0
    pontos = 1000


    print("Qual o nível de dificuldade?")
    print("(1) Fácil (2) Médio (3) Difícil")


    nivel = int(input("Defina o nível: "))


    if (nivel == 1):
        total_de_tentativas = 20
    elif (nivel == 2):
        total_de_tentativas = 10
```

```
else:
    total_de_tentativas = 5

for rodada in range(1, total_de_tentativas + 1):
    print("Tentativa {} de {}".format(rodada, total_de_tentativas))
    chute_str = input("Digite um número entre 1 e 100: ")
    print("Você digitou: ", chute_str)
    chute = int(chute_str)

    if (chute < 1 or chute > 100):
        print("Você deve digitar um número entre 1 e 100!")
        continue

    acertou = numero_secreto == chute
    maior = chute > numero_secreto
    menor = chute < numero_secreto

    if (acertou):
        print("Você acertou e fez {} pontos!".format(pontos))
        break
    else:
        if (maior):
            print("Você errou! O seu chute foi maior que o número secreto.")
        elif (menor):
            print("Você errou! O seu chute foi menor que o número secreto.")
        pontos_perdidos = abs(numero_secreto - chute)
        pontos = pontos - pontos_perdidos

print("Fim do jogo")
```

Chamando as funções

Agora precisamos chamar essas duas funções dentro do arquivo jogos.py, utilizando o nome do módulo e chamando a sua função:

```
# jogos.py
```

```
import forca
import adivinhacao

print("**********************************")
print("*******Escolha o seu jogo!*******")
print("**********************************")

print("(1) Forca (2) Adivinhação")

jogo = int(input("Qual jogo? "))

if (jogo == 1):
    print("Jogando forca")
    forca.jogar()
elif (jogo == 2):
    print("Jogando adivinhação")
    adivinhacao.jogar()
```

Podemos executar novamente o jogos.py e observar a sua saída:

```
**********************************
*******Escolha o seu jogo!*******
**********************************
(1) Forca (2) Adivinhação
Qual jogo?
```

Não aparecem mais as saídas dos jogos e sim a saída do próprio jogos.py. Os arquivos foram importados mas não foram executados, somente lidos. Eles só serão executados quando suas respectivas funções forem chamadas.
Com isso modularizamos o nosso código, temos um arquivo para cada jogo e um principal, que disponibiliza os jogos para o usuário. Podemos testar a execução do arquivo jogos.py pelo terminal também:

```
python3 jogos.py
```

E o jogo diretamente, será que conseguimos jogar? Se executarmos o arquivo adivinhacao.py, por exemplo:

```
python3 adivinhacao.py
```

Nada acontece! Ou seja, agora só conseguimos jogar os nossos jogos através do arquivo jogos.py. Vamos então tentar resolver isso no próximo vídeo, até lá!

Não conseguimos mais jogar diretamente cada jogo porque o seu próprio arquivo não chama a sua função **jogar()**. Então, depois da função, vamos chamá-la:

```
# adivinhacao.py

import random

def jogar():
    # código omitido

jogar()
```

Isso resolve o problema de jogar o jogo diretamente mas voltamos ao problema do vídeo anterior! Ao executarmos o arquivo **jogos.py**, como o próprio arquivo **adivinhacao.py** chama a função **jogar()**, ela será executada sem que queiramos isso.
Precisamos dar um jeito para que, quando executarmos o jogo de adivinhação diretamente, a função **jogar()** deve ser chamada, mas quando só o importamos, não queremos que a função seja chamada.

Programa principal vs Programa importado

Quando rodamos diretamente um arquivo no Python, ele internamente cria uma variável e a preenche. E através dessa variável podemos fazer uma consulta, pois se ela estiver preenchida, significa que o arquivo foi executado diretamente, mas se a variável não estiver preenchida, então significa que o arquivo só foi importado.
Essa variável é a **__name__**, e ela é preenchida com o valor **__main__** caso o arquivo seja executado diretamente. Vamos então fazer **if** para verificar se ela está preenchida ou não:

```
# adivinhacao.py

import random

def jogar():
    # código omitido

if (__name__ == "__main__"):
    jogar()
```

Podemos agora testar os dois casos, executar o arquivo diretamente e executar o arquivo **jogos.py**. Os dois estão funcionando, exatamente como queríamos. Falta fazermos a mesma coisa com o jogo da forca:

```
# forca.py


def jogar():
    # código omitido


if (__name__ == "__main__"):
    jogar()
```

E com o arquivo **jogos.py**, colocando o seu código dentro da função **escolhe_jogo()** e chamando-a caso o programa seja o programa principal:

```
import forca
import adivinhacao


def escolhe_jogo():
    print("**********************************")
    print("*******Escolha o seu jogo!*******")
    print("**********************************")

    print("(1) Forca (2) Adivinhação")

    jogo = int(input("Qual jogo? "))

    if (jogo == 1):
        print("Jogando forca")
        forca.jogar()
    elif (jogo == 2):
        print("Jogando adivinhação")
        adivinhacao.jogar()


if (__name__ == "__main__"):
    escolhe_jogo()
```

Com isso vimos como diferenciar se o programa é o principal ou não, se ele está sendo executado diretamente ou só sendo importado. Na hora de importar um arquivo, ele lê o código da função, mas não o executa, pois ele não é o arquivo principal

Preparando adivinhacao.py

Vamos transformar o nosso arquivo adivinhacao.py em um verdadeiro módulo, que pode ser importado ou executado diretamente.
Para tal, coloque todo o código, menos o import, dentro de uma função com o nome jogar:

```
# adivinhacao.py

import random

def jogar():
    # código omitido
```

No fim do arquivo, adicione o if que garante a execução como programa principal:

```
if(__name__ == "__main__"):
    jogar()
```

Pronto, o adivinhacao.py já foi modularizado!

Preparando o jogo de Forca

Para podermos testar corretamente a importação, vamos preparar o arquivo forca.py. No PyCharm, dentro do seu projeto jogos, crie um novo arquivo forca.py.
Nele, adicione a função jogar e o if que testa a variável __name__:

```
# forca.py

def jogar():
    print("*********************************")
    print("***Bem vindo ao jogo da Forca!***")
    print("*********************************")

    print("Fim do jogo")

if(__name__ == "__main__"):
    jogar()
```

Escolha do jogo

Para podermos escolher entre o jogo de forca e o jogo de adivinhação, crie mais um arquivo dentro do projeto jogos. Chame o arquivo de jogos.py.

Adicione o código abaixo no arquivo, para importar os módulos (`forca` e `adivinhacao`) e implementar a lógica de escolher o jogo:

```
import forca
import adivinhacao

print("**********************************")
print("*******Escolha o seu jogo!*******")
print("**********************************")

print("(1) Forca (2) Adivinhação")

jogo = int(input("Qual jogo? "))

if (jogo == 1):
    print("Jogando forca")
    forca.jogar()
elif (jogo == 2):
    print("Jogando adivinhação")
    adivinhacao.jogar()
```

Repare que chamamos a função `jogar` de cada módulo.

Executando o jogo

Dentro do PyCharm, rode o arquivo jogos.py. Fique atento para pegar possíveis erros de sintaxe.