

Sobre Este Curso

Público Alvo

Programadores que têm como objetivo aprender a base para atuar como Desenvolvedor Python.

Pré-Requisitos

Conhecimentos em Lógica de Programação.

Índice

SOBRE ESTE CURSO	I
PÚBLICO ALVO	I
PRÉ-REQUISITOS	I
ÍNDICE.....	I
COPYRIGHT	III
EQUIPE.....	III
HISTÓRICO DAS EDIÇÕES	III
CAPÍTULO 01 – O QUE É PYTHON?	4
CAPÍTULO 02 – FAZENDO O DOWNLOAD DO PYTHON	5
CAPÍTULO 03 – FAZENDO DOWNLOAD DO PYCHARM.....	7
CAPÍTULO 04 – CRIANDO UM PROJETO COM PYCHARM	9
CAPÍTULO 05 – TIPOS DE DADOS COM PYTHON.....	15
CONVERSÕES DE DADOS COM PYTHON	18
CAPÍTULO 06 – OPERADORES.....	19
CAPÍTULO 07 – INPUT E OUTPUT	26
CAPÍTULO 08 – CRIANDO MÉTODOS	29
CAPÍTULO 09 – FUNÇÕES DE MANIPULAÇÃO DE TEXTO COM PYTHON	32
CAPÍTULO 10 – FUNÇÕES MATEMÁTICAS.....	34
CAPÍTULO 11 – COMANDOS CONDICIONAIS	36
CAPÍTULO 12 – LAÇOS DE REPETIÇÃO	38
CAPÍTULO 13 – VARIÁVEIS INDEXADAS UNIDIMENSIONAIS	41
CAPÍTULO 14 – PYTHON - PROGRAMAÇÃO ORIENTADA A OBJETOS	48
CAPÍTULO 15 – CONECTANDO AO BANCO DE DADOS MYSQL	58
CAPÍTULO 16 – UTILIZANDO FILTER NO PYTHON.....	65

	Anotações

Anotações	

Copyright

As informações contidas neste material se referem ao curso de **Python** e estão sujeitas as alterações sem comunicação prévia, não representando um compromisso por parte do autor em atualização automática de futuras versões.

A **Apex** não será responsável por quaisquer erros ou por danos acidentais ou consequenciais relacionados com o fornecimento, desempenho, ou uso desta apostila ou os exemplos contidos aqui.

Os exemplos de empresas, organizações, produtos, nomes de domínio, endereços de e-mail, logotipos, pessoas, lugares e eventos aqui representados são fictícios. Nenhuma associação a empresas, organizações, produtos, nomes de domínio, endereços de e-mail, logotipos, pessoas, lugares ou eventos reais é intencional ou deve ser inferida.

A reprodução, adaptação, ou tradução deste manual mesmo que parcial, para qualquer finalidade é proibida sem autorização prévia por escrito da **Apex**, exceto as permitidas sob as leis de direito autoral.

Equipe

Conteúdos

- Gustavo Rosauro

Diagramação

- Elizabeth Pereira

Revisão

- Fernanda Pereira

Histórico das Edições

Edição	Idioma	Edição
1ª	Português	Janeiro de 2020

© Copyright 2020 **Apex**. Desenvolvido por Gustavo Rosauro e licenciado para Apex Ensino

	Anotações

Capítulo 01 – O que é Python?



Python é uma linguagem dinâmica, interpretada, robusta, multiplataforma, multi-paradigma. (Orientada à objetos, funcional, refletiva e imperativa).

Está preparada para rodar em JVM e .NET.

É uma linguagem livre (até para projetos comerciais) e hoje pode-se programar com Python softwares para desktop, web, e mobile.

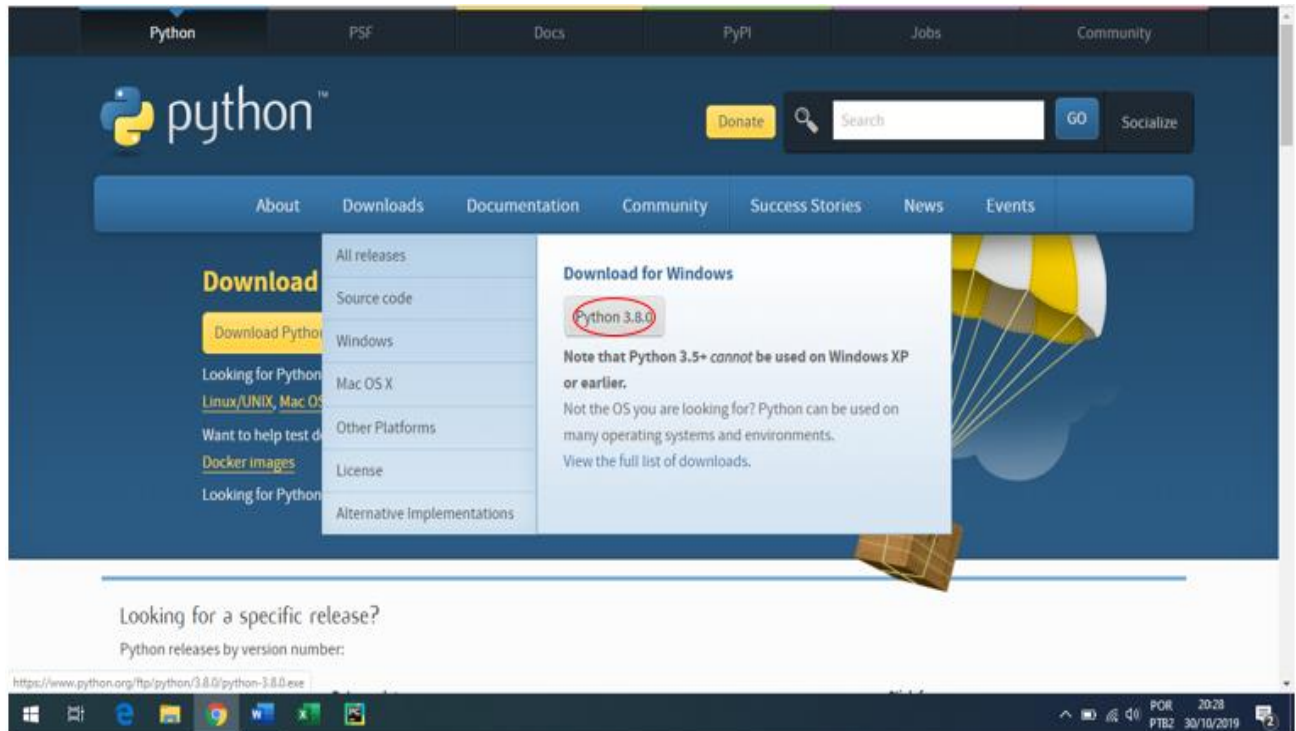
A Linguagem Python começou a ser criada no final da década de 80 por Guido Van Rossum, um importante programador e pesquisador do Instituto Nacional de Matemática e Ciência da Computação da Holanda. Seu lançamento ocorreu em 1991.

O holandês é um grande adepto da melhoria contínua, pois o Python só surgiu devido à necessidade do aprimoramento de outro software de programação usado na instituição até então chamado de ABC (não confunda com curva ABC).

O processo de melhoria buscava torná-lo mais ágil e tinha como objetivo otimizar a sua capacidade interação com um sistema operacional de interesse na época, com redução do tempo de execução dos programas para garantir maior eficiência do conjunto.

Anotações	

Capítulo 02 – Fazendo o Download do Python



Primeiramente, precisamos baixar o ambiente de desenvolvimento integrado do Python, chamado de IDLE.

Esse é local onde escreveremos nosso programa.

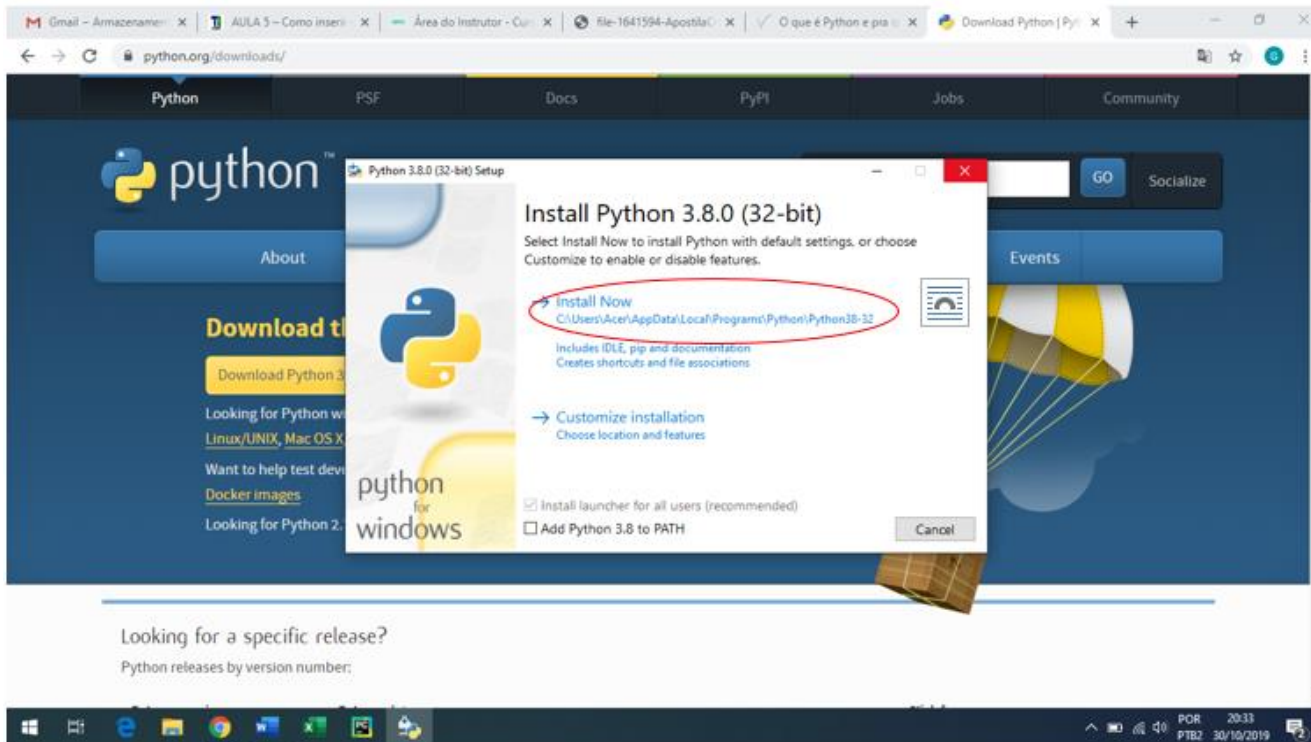
Para isso, basta acessar a página oficial da Python Software Foundation, ir até a aba “Downloads” e clicar em Python.

Nesse tutorial, vamos usar a plataforma Windows. Quanto ao uso desse software, podemos levantar duas vantagens. Primeira, ele é de licença de uso público, ou seja, é gratuito. E segunda, é multiplataforma, portanto, funciona em vários sistemas operacionais diferentes e até mesmo em aparelhos móveis!

Abra o arquivo de instalação baixado.

Em seguida, clique em “Install Now” e aguarde a mágica acontecer!

	Anotações



Anotações	

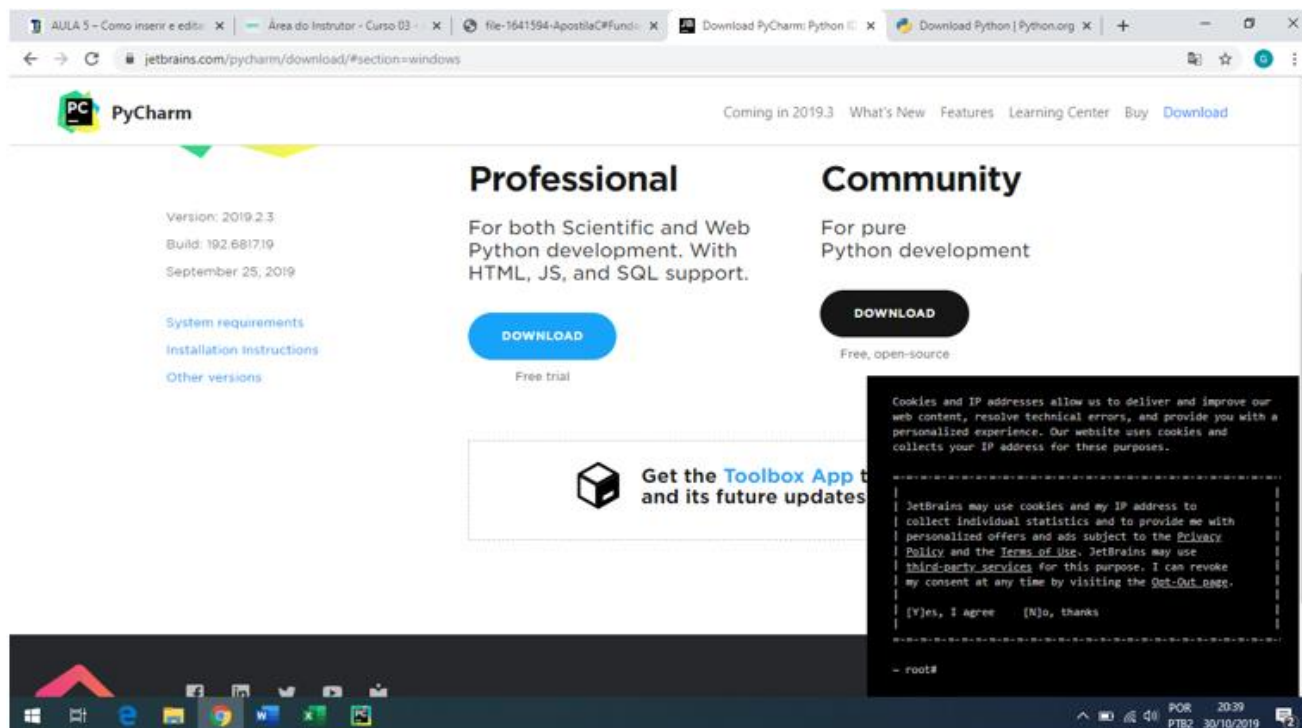
Capítulo 03 – Fazendo Download do PyCharm



Para que possamos executar os comandos do Python, iremos utilizar um editor de texto que é próprio para que possamos executar os nossos comandos nesta linguagem. Ele pode ser instalado no site

<https://www.jetbrains.com/pycharm/>

	Anotações



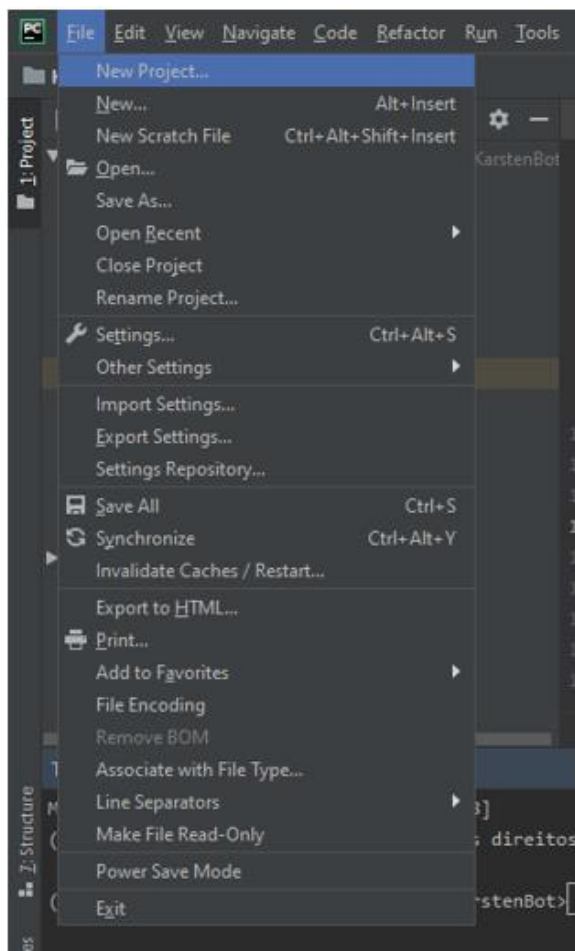
The screenshot shows the JetBrains PyCharm download page. The browser tabs include 'AULA 3 - Como inserir e editar', 'Área do Instrutor - Curso 03', 'file-1641594-ApostilaC#Fund...', 'Download PyCharm: Python', and 'Download Python | Python.org'. The URL is 'jetbrains.com/pycharm/download/#section=windows'. The page features the PyCharm logo and navigation links: 'Coming in 2019.3', 'What's New', 'Features', 'Learning Center', 'Buy', and 'Download'. The main content is divided into two sections: 'Professional' and 'Community'. The 'Professional' section describes it as 'For both Scientific and Web Python development. With HTML, JS, and SQL support.' and offers a 'Free trial' download. The 'Community' section describes it as 'For pure Python development' and offers a 'Free, open-source' download. A 'Get the Toolbox App' banner is also present. A cookie consent banner is visible at the bottom right, asking for permission to use cookies and IP addresses for a personalized experience.

Após clicado no botão download, agora iremos escolher a opção community, que é uma opção para estudantes e gratuita.

Anotações	

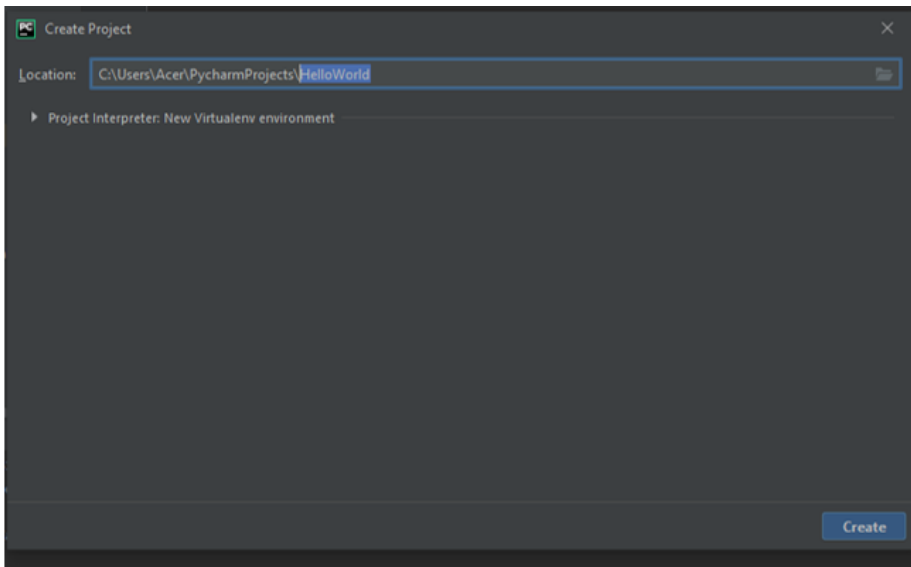
Capítulo 04 – Criando um projeto com PyCharm

Agora que já instalamos nossa IDE, vamos começar a entender como é a Linguagem Python. Abra o Pycharm e clique no canto superior esquerdo. Escolha a opção New Project.

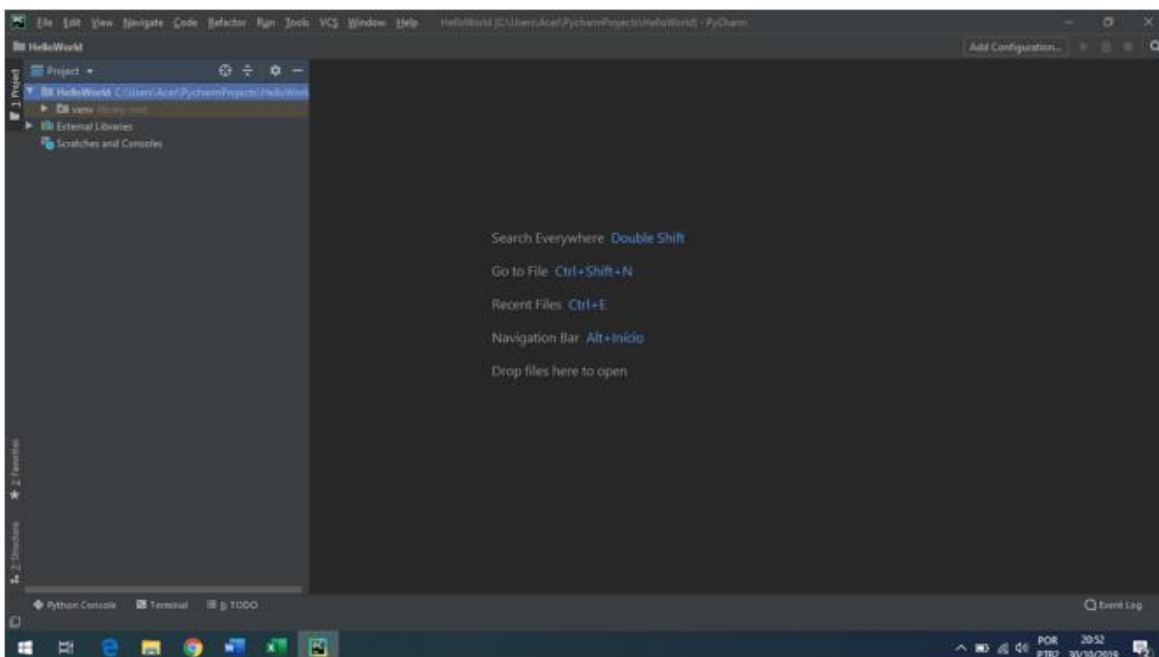


Após escolhida essa opção, agora iremos colocar o nome do projeto que vamos começar a criar. Conforme demonstra a imagem abaixo:

	Anotações



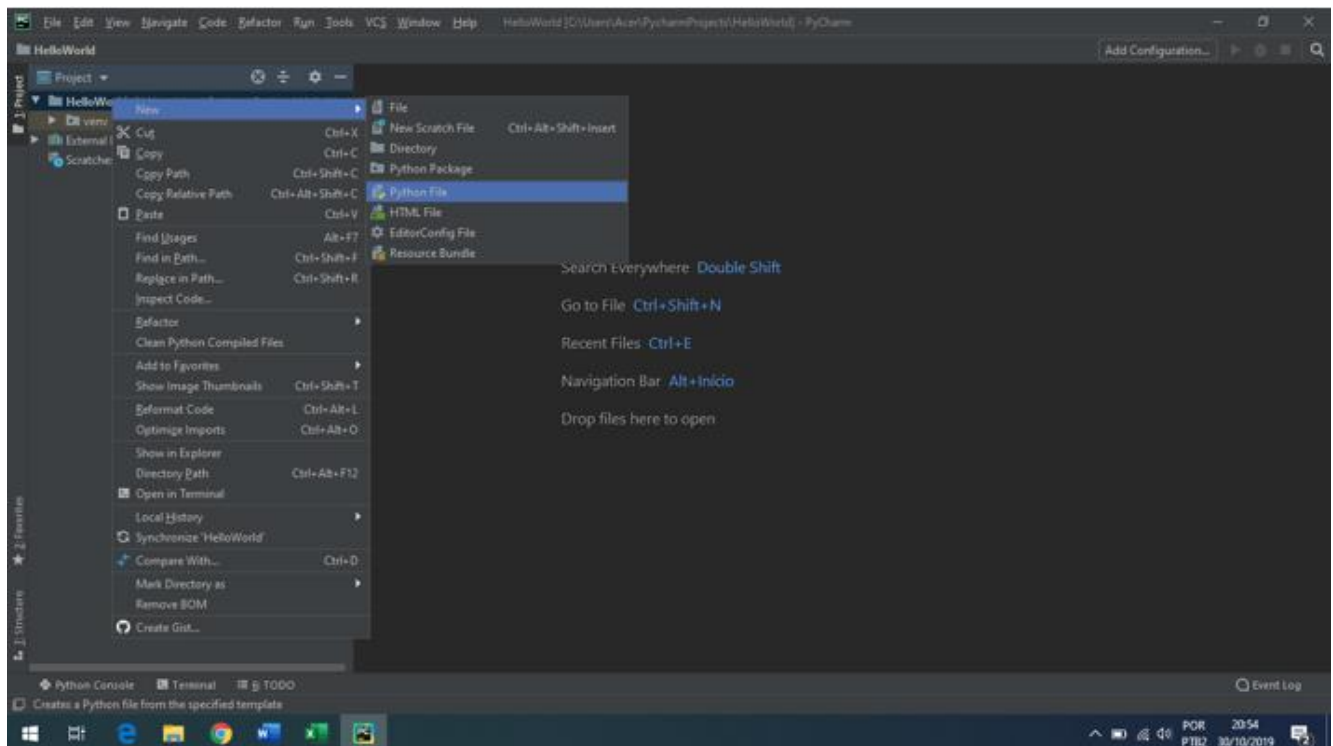
Nosso projeto será criado com uma pasta chamada *helloworld* e dentro dela teremos uma outra pasta chamada *venv* que contém as bibliotecas para que possamos executar nosso projeto.



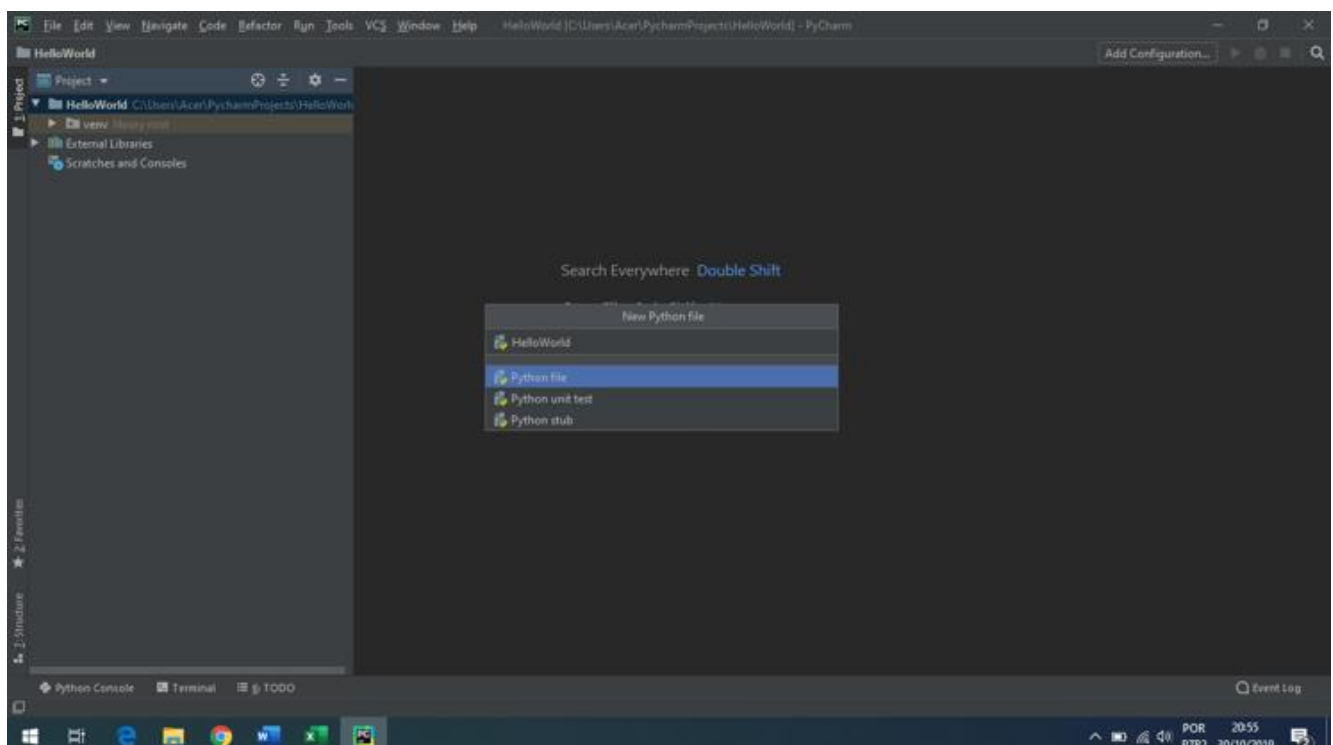
Agora iremos criar um arquivo *python* para executar a nossa lógica.

Para fazer isso, clicamos com o botão direito em cima da pasta e escolhemos a opção *new -> file -> python file*.

Anotações	



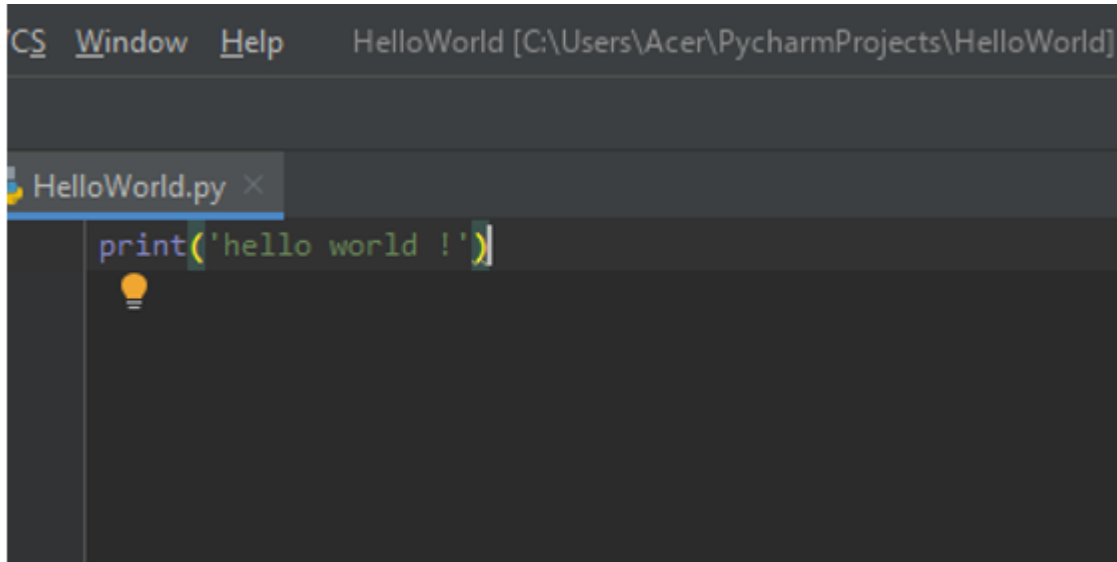
E iremos colocar o nome helloWorld e clicar em ok.



	Anotações

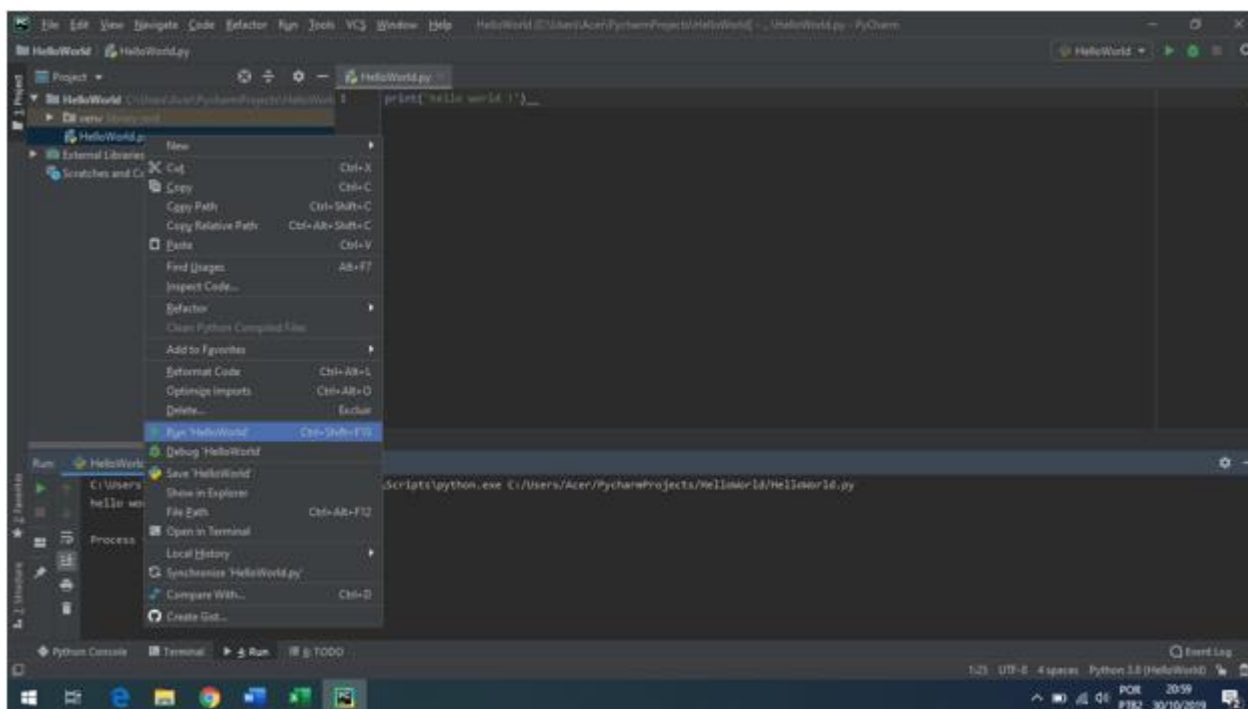
Para nossa lógica, iremos escrever hello world.

Para que possamos escrever esse resultado com python utilizaremos o comando print.



```
CS Window Help HelloWorld [C:\Users\Acer\PycharmProjects\HelloWorld]
HelloWorld.py x
print('hello world !')
```

Vamos clicar com o botão direito em cima da nossa classe escolher a opção run.



E iremos ver em nosso console o seguinte resultado:

Anotações	

```

Run: HelloWorld x
C:\Users\Acer\PycharmProjects\HelloWorld\venv\Scripts\python.exe
hello world !
Process finished with exit code 0
  
```

Para interagir com o Python nesse primeiro momento utilizaremos o comando *input* e iremos armazenar seu valor em uma variável.

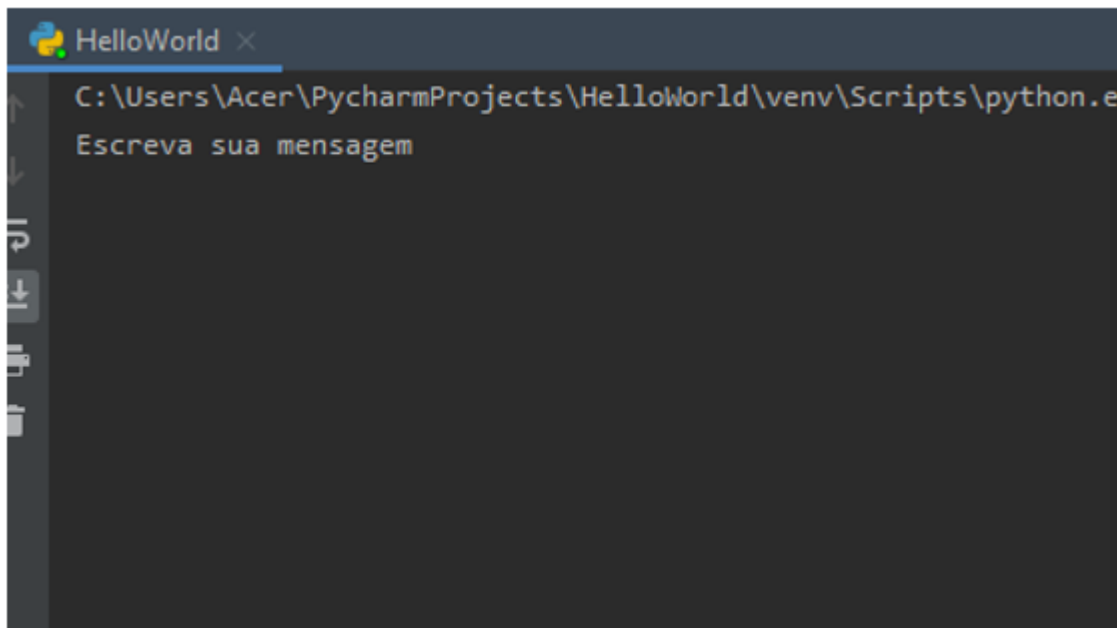
As variáveis no Python não são fortemente tipadas, elas recebem o valor que está depois do sinal de igual, conforme demonstra a imagem abaixo:

```

1 mensagem = input('Escreva sua mensagem')
2
3 print(mensagem)
  
```

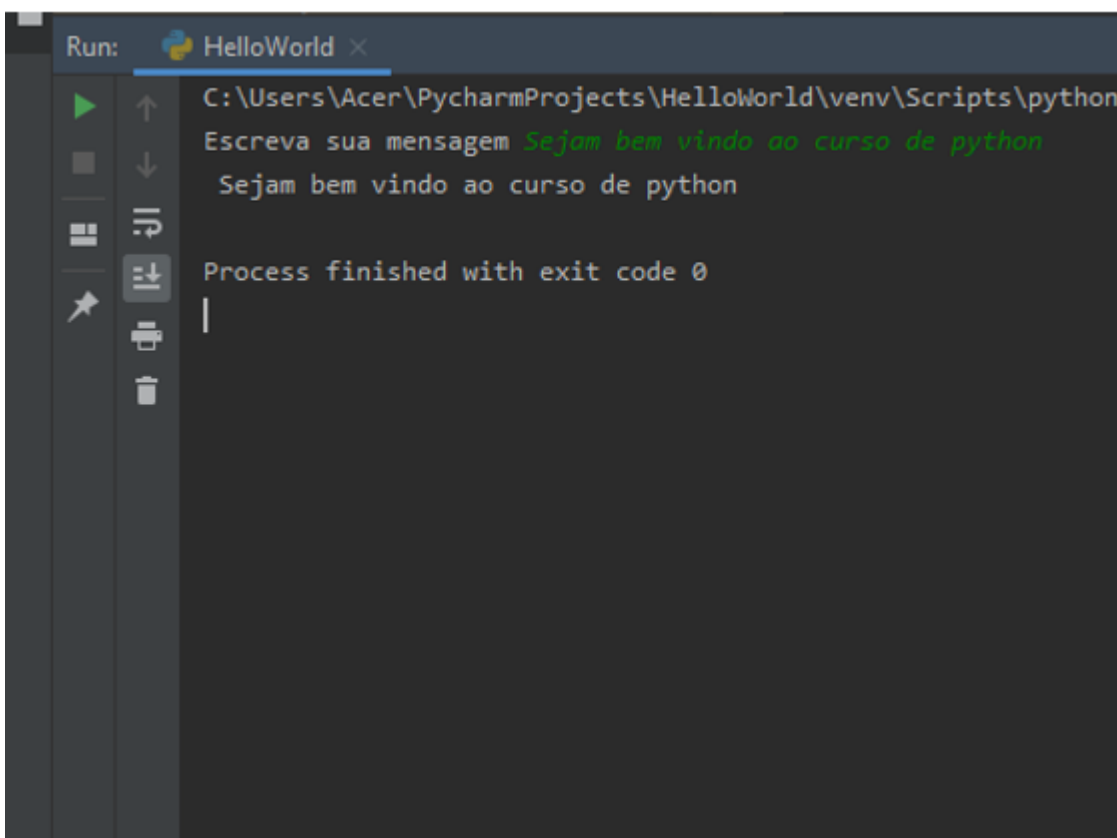
Nossa variável nesse exemplo é a *mensagem*, que recebe o *input*, um comando que espera a entrada de dados do usuário em um formato de texto.

	Anotações



```

HelloWorld x
C:\Users\Acer\PycharmProjects\HelloWorld\venv\Scripts\python.exe
Escreva sua mensagem
  
```



```

Run: HelloWorld x
C:\Users\Acer\PycharmProjects\HelloWorld\venv\Scripts\python.exe
Escreva sua mensagem Sejam bem vindo ao curso de python
Sejam bem vindo ao curso de python

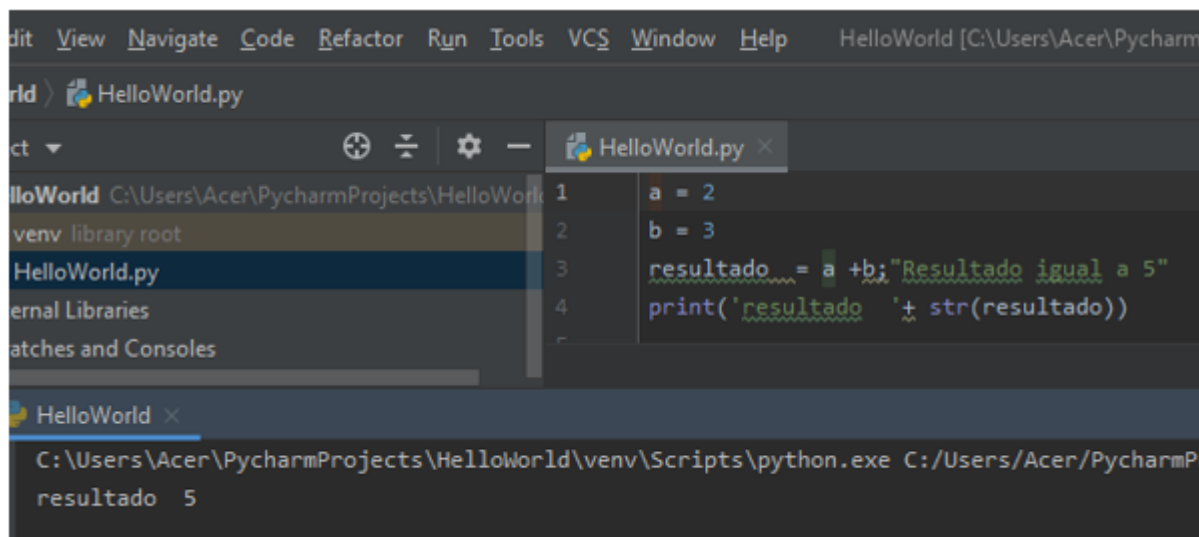
Process finished with exit code 0
  
```

Anotações	

Capítulo 05 – Tipos de dados com Python

Type	Example
• Numeric: Integer, Float	x = 10 x = 1.0
• String	x= 'Mike'
• Boolean	y = True x = False
• List	my_list = [10, 20, 30]

Numeric são os tipos de dados que nos possibilitam fazer as quatro operações matemáticas.



```

1 a = 2
2 b = 3
3 resultado = a + b; "Resultado igual a 5"
4 print('resultado ' + str(resultado))

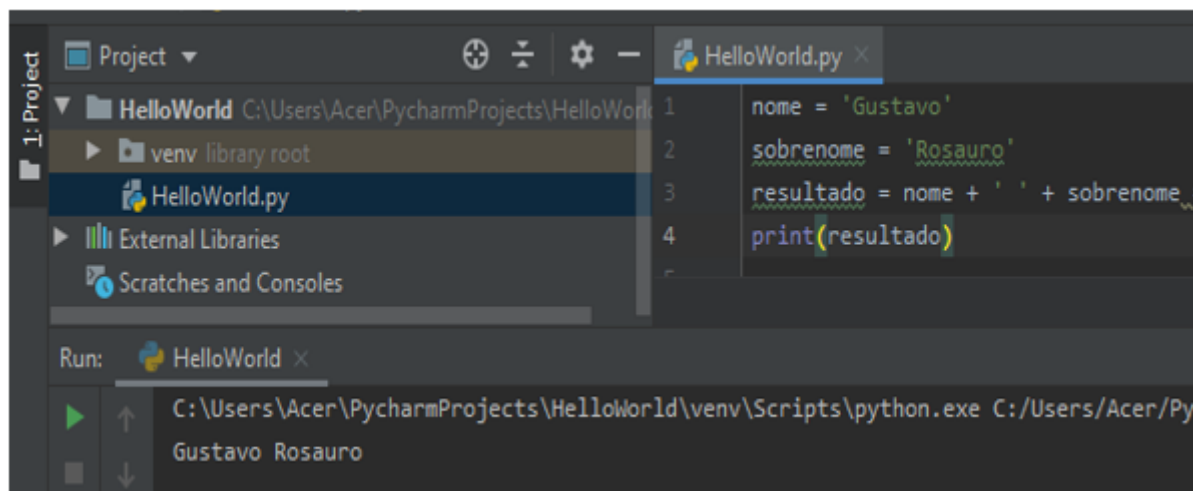
```

Output: resultado 5

String: são os tipos de dados no formato de texto.

Como no exemplo quando para utilizarmos string utilizamos aspas simples ou duplas.

	Anotações



```

1 nome = 'Gustavo'
2 sobrenome = 'Rosauro'
3 resultado = nome + ' ' + sobrenome
4 print(resultado)

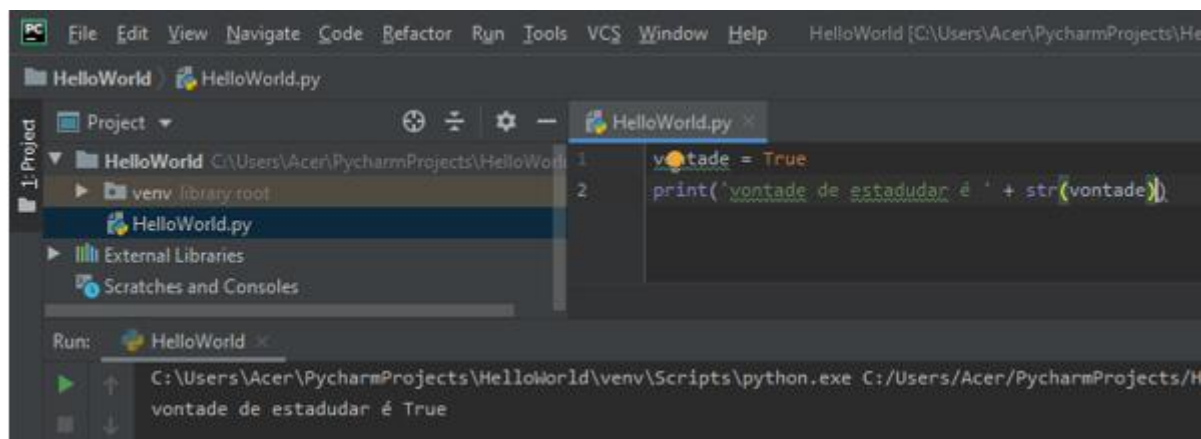
```

Run: HelloWorld x

C:\Users\Acer\PycharmProjects\HelloWorld\venv\Scripts\python.exe C:/Users/Acer/PycharmProjects/HelloWorld/HelloWorld.py

Gustavo Rosauro

Boolean: é o tipo de dados primitivos, que utiliza a sentença verdadeira ou falso como resultado ou 1 ou 0.



```

1 vontade = True
2 print('vontade de estadudar é ' + str(vontade))

```

Run: HelloWorld x

C:\Users\Acer\PycharmProjects\HelloWorld\venv\Scripts\python.exe C:/Users/Acer/PycharmProjects/HelloWorld/HelloWorld.py

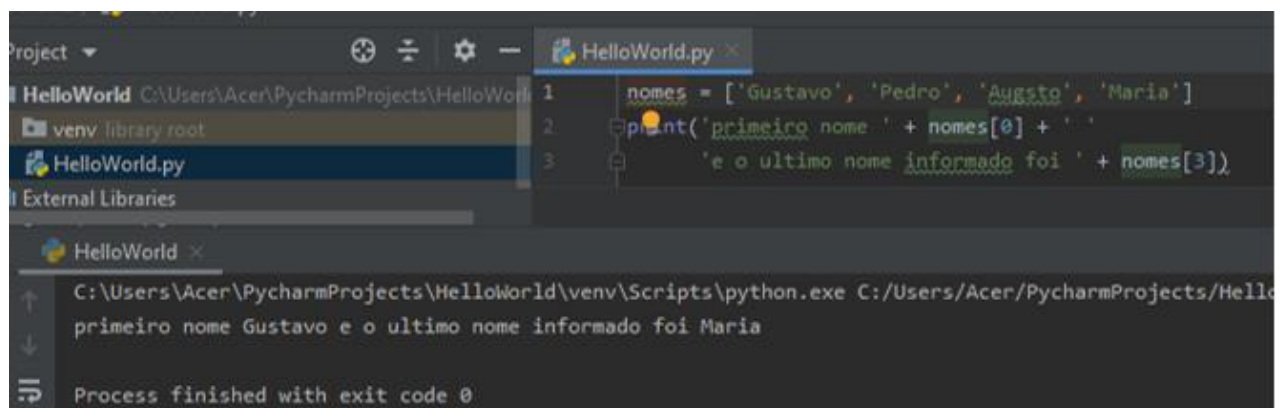
vontade de estadudar é True

O tipo List é o array que nada mais é do que um conjunto de dados do mesmo tipo, por exemplo, vamos supor que eu tenha uma lista de nomes dos meus clientes.

Como eu iria salvar esses nomes no mesmo local?

Não seria viável criar uma variável para cada um, então utilizaríamos uma lista.

Anotações	



The screenshot shows the PyCharm IDE interface. The top pane displays a Python file named `HelloWorld.py` with the following code:

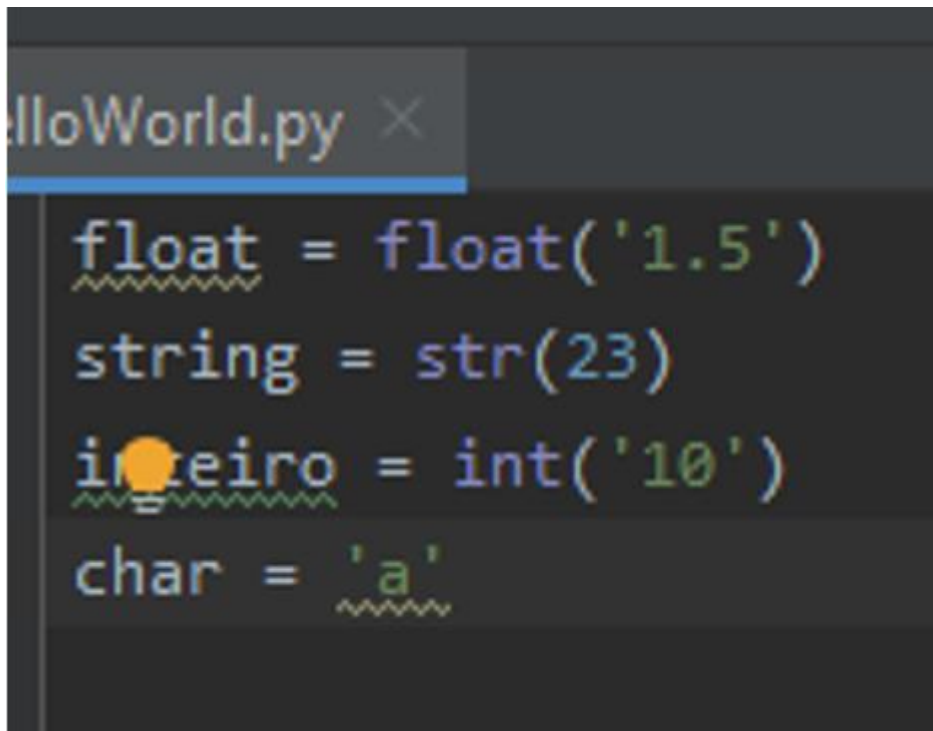
```
1 nomes = ['Gustavo', 'Pedro', 'Augusto', 'Maria']
2 print('primeiro nome ' + nomes[0] + ' '
3       'e o ultimo nome informado foi ' + nomes[3])
```

The bottom pane shows the output of the script execution:

```
C:\Users\Acer\PycharmProjects\HelloWorld\venv\Scripts\python.exe C:/Users/Acer/PycharmProjects/Hell
primeiro nome Gustavo e o ultimo nome informado foi Maria
Process finished with exit code 0
```

	Anotações

Conversões de dados com Python



```
helloWorld.py ×  
  
float = float('1.5')  
string = str(23)  
inteiro = int('10')  
char = 'a'
```

Nessa imagem mostramos como converter string para float, int para string e string para inteiro.

E logo abaixo a declaração de um char que nada mais é que a declaração de um character.

Anotações	

Capítulo 06 – Operadores

Quando pensamos em Python os operadores aritméticos podem ser entendidos basicamente como:

Operadores	Função
+	Adição
-	Subtração
*	Multiplicação
/	Divisão
%	Resto/Módulo

Adição

a = 3

b = 4

c = a + b

#Resultado c = 7.

Subtração

a = 3

b = 4

c = a - b

#Resultado c = -1.

Multiplicação

a = 3

b = 4

c = a * b

#Resultado c = 12.

	Anotações

Divisão

```
a = 11;
```

```
b = 2;
```

```
c = a / b;
```

```
#Resultado c = 5.
```

Resto/ Módulo

```
a = 11;
```

```
b = 2;
```

```
c = a / b;
```

```
#Resultado c = 1.
```

Comparação

Operadores de Comparação

Operador	Significado
==	Igualdade
>	Maior
<	Menor
<=	Menor Igual
>=	Maior igual
!=	Diferente

```
A = 5
```

```
B = 3
```

```
A == B # Falso
```

```
A > B # Verdadeiro
```

```
A < B # Falso
```

Anotações	

A <= B #Falso

A >= B #Verdadeiro

A != B # Verdadeiro

Lógicos

Operadores lógicos

AND = E

OR = OU

```
a = 1
b = 3
primeiroTeste = a > 3 and b < 10 # primeiroTeste = false
```

Para que um teste lógico seja verdadeiro perante dois testes ambos tem de ser verdadeiros.

```
a = 1
b = 3
primeiroTeste = a > 3 or b < 10 # primeiroTeste = true
```

Atribuição

Operadores de Atribuição

Operador	Significado
=	Atribuição Simples
+=	Atribuição Aditiva
-=	Atribuição Subtrativa
*=	Atribuição Multiplicativa
/=	Atribuição de divisão
%=	Atribuição Modular
++	Adição de 1 unidade
--	Subtração de 1 unidade

	Anotações

Simples

```
simp = 'simples'
```

Aditiva

```
a = 3
```

```
a += 1
```

```
#a = 4
```

Subtrativa

```
a = 3
```

```
a -= 1
```

```
#a = 2
```

Multiplicativa

```
a = 3
```

```
a *= 4
```

```
#a = 12
```

Divisão

```
a = 8
```

```
a /= 2
```

```
#a = 4
```

Anotações	

Modular

`a = 11`

`a %= 2`

`#a = 1`

Adição de uma Unidade

`a = 11`

`a++`

`#a=12`

Subtração de uma Unidade

`a = 11`

`a—`

`#a=10`

Concatenação

Operador de concatenação

Operador	Significado
+	Concatena caracteres e cadeias de caracteres

O operador de concatenação tem como objetivo juntar, concatenar dois ou mais caracteres.

Exemplo

	Anotações

```
nome = "Gustavo"  
sobrenome = "Rosauro"  
resultado = nome + ' ' + sobrenome #Gustavo Rosauro
```

Operadores Ternários

Já vimos anteriormente a existência de operadores lógicos, aritméticos, de comparação, de atribuição e concatenação.

Todos esses operadores funcionam semelhantemente a outras linguagens, porém os Operadores Ternários são restritos a algumas linguagens de programação. Felizmente o python é uma delas.

Os operadores são formados por três Operandos, por isso o nome, e são separados por dois caracteres o 'if' e o 'else', conforme no exemplo abaixo

valor_se_verdadeiro if condição else valor se falso

Dessa maneira fazemos uma pergunta da qual retorne um valor Verdadeiro ou Falso.

Caso o valor seja verdadeiro, retornamos o valor que estará na posição 'valor_se_verdadeiro' posicionado antes do 'if'.

Caso ele seja um valor falso, retornamos o valor que estará na posição 'valor_se_falso' posicionado após o 'else'

Os operadores ternários têm como objetivo resolver de uma maneira mais simples uma expressão lógica.

Exemplo:

```
a = 2  
b = 2  
resultado = "verdadeiro" if a == b else "falso"  
print(resultado) #verdadeiro
```

Anotações	

	Anotações

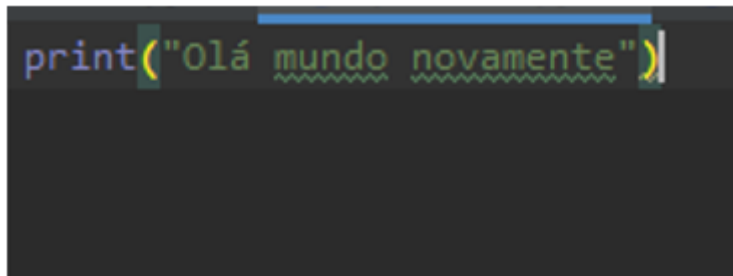
Capítulo 07 – Input e Output

Assim como na Lógica de Programação, em Python também executamos comandos que nos fornecem informações, seja escrevendo na tela ou em um arquivo, seja pedindo para que passamos algum parâmetro para o programa.

Comandos de saída (Output)

Quando pensamos em comando de saída, lembramos de um comando que nos dirá alguma coisa. Sendo ele responsável por exibir a informação que estamos querendo que seja exibida.

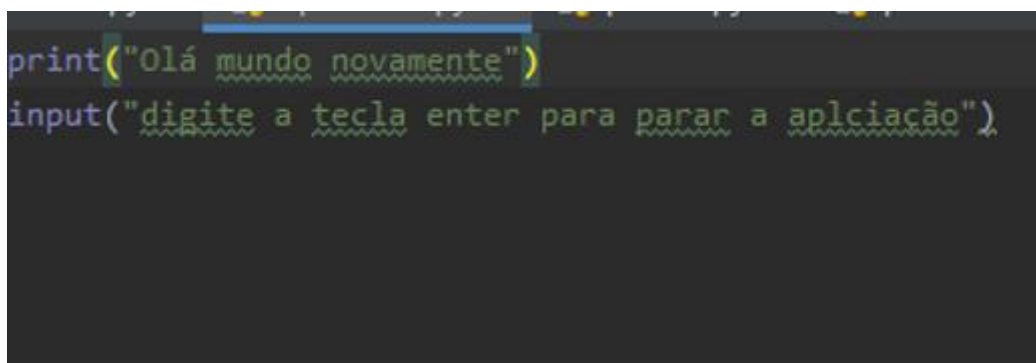
Para realizar tal tarefa em Python utilizaremos o comando “print()”



```
print("Olá mundo novamente")
```

Comandos de entrada (Input)

Seja para fazer com que o código espere o parâmetro para terminar de executar o programa, ou seja para realmente passar como parâmetro algum valor para o programa, devemos utilizar um método de input, que fará com que o programa espere receber algum valor, conforme o exemplo utilizando “input()”, que lê tudo o que você digitar e só termina quando você apertar a tecla ENTER.



```
print("Olá mundo novamente")
input("digite a tecla enter para parar a aplicação")
```

Anotações	

Exemplo interagindo com o usuário

#string

```
texto = input("digite um texto qualquer")
print("texto digitado: "+texto)
```

#int

```
numero = int(input("digite um numero"))
print(numero + numero)
```

Todos os tipos de dados do Python permitem conversão para outros tipos, lembrando que devem conter o conteúdo correto. Por exemplo, não é possível converter “meu nome é” em int.

	Anotações

Anotações	

Capítulo 08 – Criando Métodos

Toda a linguagem de programação que se preze permite a criação de métodos e funções pelo programador.

Para criar um método no Python temos de obedecer a seguinte estrutura:

```
def nome_do_metodo(paremetros):
```

```
    print('o que tiver com um espaço da margem abaixo do def ele já entende como parte da função')
```

Como por exemplo:

```
texto = input('digite um texto')
def Escrever(texto):
    print(texto)

Escrever(texto)
```

Os métodos por padrão, quando queremos utilizar na mesma página, temos de criar em cima para podermos fazer a chamada desse método.

Quando utilizamos Métodos evitamos a reescrita do código.

Por exemplo, vamos imaginar que tenhamos uma rotina que calcula a média da nota de um aluno, mas temos mais de um aluno, como criar essa rotina sem precisar reescrever a fórmula:

```
def media(nota1,nota2,nota3):
    return str((nota1+nota2+nota3)/3)
print("Primeiro Aluno "+ media(10,8,7))
print("Segundo Aluno "+media(7,7,7))
print("Terceiro Aluno "+media(5,9,6))
```

	Anotações

Exercícios

- 1- Crie uma aplicação que calcule o IMC do indivíduo, pesquise a fórmula e aplique.
- 2- Porque devemos utilizar métodos?
- 3- O que são inputs e outputs?

Anotações	

	Anotações

Capítulo 09 – Funções de manipulação de texto com Python

Funções de manipulação de texto tem como objetivo preparar o texto de uma determinada ação ou exibição.

Dentro do Python existem inúmeras funções. Nesse capítulo veremos quais são as mais utilizadas no dia a dia de um programador.

Replace (texto antigo, texto novo)

A função Replace da classe string é provavelmente a função de texto mais utilizada na programação. Tem como objetivo substituir determinado caractere ou texto dentro de uma variável string por outro texto caractere.

Exemplo:

```
texto = "escrevendo texto ANTIGO"
textoNovo = texto.replace("ANTIGO", "NOVO")
print(textoNovo)
```

Texto Substring [inicio_index:fim_index]

A função substring tem como objetivo conseguir separar somente uma determinada parte do texto contido em outra variável. Para essa função ocorrer de maneira correta, devemos passar o index (posição do caractere) que gostaríamos de começar até qual índice gostaríamos que ele mostrasse.

Lembrando que em Python os índices começam com 0.

Exemplo:

```
nome = 'gustavo rosauro'
print(nome[0:1]) #g
```

Lower(string)

A função lower, assim como o nome já diz, tem como objetivo diminuir a string, isso significa tornar as letras que são MAIÚSCULAS em minúsculas.

Anotações	

```
nome = 'GUSTAVO ROSAURO'  
print(nome.lower()) #gustavo rosauro
```

Upper(string)

A função upper, assim como o nome já diz, tem como objetivo aumentar a string, isso significa tornar as letras que são minúsculas em maiúsculas.

```
nome = 'gustavo rosauro'  
print(nome.upper()) #GUSTAVO ROSAURO
```

	Anotações

Capítulo 10 – Funções matemáticas

Pow(x,y)

Essa função tem como objetivo retornar um número elevado a potência de outro número.

```
a = 5
print(pow(a,3))
```

O resultado que irá aparecer na tela será 135 pois nesse caso é 5 elevado a 3.

Round(X,Y)

Essa função arredonda o número X para que ele fique com Y casas decimais. Quando o valor de Y não for passado, considera-se 0 casas decimais, ou seja, um valor inteiro.

```
a = 2.34242
b = round(a,2)
print(b) #2.34
```

Math.Sqrt(X)

Função responsável por retornar a raiz quadrada de um número.

Exemplo:

Anotações	

```
import math
a = 2
b = math.sqrt(a)
print(b)
```

Exercícios

1 – Crie um programa com as seguintes funções:

- Receba três parâmetros, um Nome (nome e sobrenome), uma idade e um telefone (somente números).
- Faça uma rotina para padronizar o formato do telefone (xx)99999-9999 na tela.
- Faça uma rotina que calcule a raiz quadrada da idade do usuário e retorne para o método principal um double, que será exibido na tela.
- Faça uma rotina para que troque os espaços entre os nomes do usuários por “;” e exiba na tela.

	Anotações

Capítulo 11 – Comandos Condicionais

Estruturas de decisão

Qualquer linguagem de programação necessita de estruturas de decisão. Isso significa que uma das bases da programação é a comparação.

A programação em Python não seria indiferente e para isso existem os comandos:

if (se)

else (senão)

Comandos IF/else/ elif

Exemplo de comandos if else

```
idade = 18
if idade == 18:
    print('Você é maior de idade')
else:
    print('Você ainda não é maior de idade')
```

Utilizando elif que seria como um else porém atendendo uma condição.

Quando não atender a condição superior ele irá cair na próxima condição do elif até que uma seja verdadeira.

Caso contrário ele irá cair no else.

Anotações	

```
idade = 27
if idade > 18 and idade < 60:
    print('você é adulto')
elif idade < 18 and idade > 12:
    print('você é adolescente')
elif idade < 12:
    print('você é criança')
else:
    print('você é idoso')
```

Exercícios

- 1) Em seu projeto python desenvolva um algoritmo que receba a idade do usuário e:
 - a. Se a idade for maior ou igual a 13 e menor que 19 escreva “Adolescente”.
 - b. Se a idade for maior ou igual a 19 e menor ou igual a 60 escreva “Adulto”.
 - c. Se a idade for maior que 60 escreva “Idoso”.
 - d. Caso contrário escreva “Criança”.
- 2) No mesmo projeto, crie um novo procedimento em que pergunte ao usuário do sistema qual time ganhou a copa do mundo de futebol de 2014. Dê a ele 4 opções, dentro dessas uma deve ser a verdadeira. Caso o usuário acerte, escreva “Acertou” caso contrário “Errou”.
- 3) Utilize o mesmo projeto de IMC dos módulos anteriores e exiba o a classificação de acordo com o resultado (ex: abaixo do peso, sobrepeso, obesidade, etc).

	Anotações

Capítulo 12 – Laços de Repetição

Neste capítulo vamos conhecer os Comandos While, Comando For, Comando For com Array.

Comando While

O Comando while é o mais simples da parte de laços de repetição quando utilizamos Python. Podemos entendê-lo como um repetidor de informação.

Exemplo:

```
contador = 0
while(contador <= 10):
    print(contador)
    contador += 1
```

Comando For

O Comando For tem as mesmas utilidades de um comando while, porém o incremento da variável cuja expressão lógica está vinculada é feito dentro do próprio comando.

Exemplo:

```
for x in range(1, 11, 1):
    print(x)
```

Anotações	

Nesse exemplo, no range o primeiro número é o número inicial. O segundo número é o comparador. E o terceiro é o número que será incrementado a cada repetição.

Exemplo Avançado

No exemplo abaixo vamos relacionar diversos ensinamentos até aqui, colocando em prática e aprendendo como lidar com situações mais complexas.

Nesse exemplo relacionaremos os comandos While, For, comandos de input e output, estruturas de decisão e funções predefinidas.

O nosso objetivo é fazer um programa que leia um número e peça para o computador contar e escrever de 1 até o número digitado.

Além disso, o programa deverá perguntar se queremos fazer novamente a contagem, porém com um valor novo.

Segue o código abaixo para análise.

```
satisfeito = False
while(satisfeito == False):
    numero = int(input('até qual número devo decrementar'))
    for i in range(1, numero, 1):
        print(i)
    if input('Satisfeito? S - para Sim / N - para Não').upper() == 'S':
        satisfeito = True
```

For com Array

Existem variáveis que armazenam informações em forma de vetores e matrizes. No python para lermos cada item desse vetor utilizamos o seguinte comando: for in 'array'.

Ele automaticamente fará a conversão do nosso vetor para cada item, conforme demonstra o exemplo abaixo:

	Anotações

```

nomes = ['Gustavo', 'Juliano', 'Fernanda', 'Ralph', 'Jefferson']
for nome in nomes:
    print(nome)

for nome in nomes
operacoes x
C:\Users\Acer\PycharmProjects\HelloWorld\venv\Scripts\python.exe
Gustavo
Juliano
Fernanda
Ralph
Jefferson

```

Exercícios

- 1) Crie um novo projeto chamado “ExerciciosLacosRepeticao”. Em sua classe Program.cs desenvolva um algoritmo que receba um número e escreva-o decrescendo de 1 em 1 até o valor chegar a zero.
- 2) No mesmo projeto crie um novo método que receba do usuário números e enquanto não for passado um número PAR o programa solicite novamente. Passado o número par, calcule a metade desse número e chame-o de X. Exiba na tela X vezes “Repetição” concatenado com o número atual das X vezes.
- 3) No mesmo projeto, crie um novo procedimento em que pergunte ao usuário do sistema qual time ganhou a última copa do mundo de futebol. Dê a ele 4 opções, dentro dessas uma deve ser a verdadeira. Caso o usuário acerte, escreva “Acertou” caso contrário, “Errou”. Não pare o programa até que ele acerte a resposta.

Anotações	

Capítulo 13 – Variáveis indexadas unidimensionais

Array

O Python, assim como a maioria das linguagens de programação atuais, suporta indexação de valores de mesmo tipo em uma estrutura única, Vetor ou também chamado de Array.

Vetores em python são pelos colchetes '['']. Conforme você pode ver no exemplo abaixo.

```
arrayDeInteiros = [0, 1, 2, 3, 8, 5, 10, 7]
arrayDeTextos = ['Segunda', 'Terça', 'Quarta']
print(arrayDeInteiros)
print(arrayDeTextos)
```

Dessa maneira conseguimos criar vetores unidimensionais de tamanho fixo, onde não podemos alterar a quantidade de valores que existem dentro do vetor. Um vetor de 10 posições pode ter no máximo 10 valores.

Outra maneira de desenvolvermos a mesma ideia é estipular o tamanho do vetor na sua criação e posteriormente adicionar valores a cada posição, conforme o exemplo:

	Anotações

```
arrayDeInteiros = [0, 1, 2, 3, 4]
arrayDeInteiros[0] = 1
arrayDeInteiros[1] = 2
arrayDeInteiros[2] = 3
arrayDeInteiros[3] = 4
arrayDeInteiros[4] = 5
for x in arrayDeInteiros:
    print(x)
```

Trabalhando com listas no Python

Em linguagens de programação mais antigas, como o C, existiam momentos em que gostaríamos de crescer o tamanho de nossos arrays, pois precisávamos de maior quantidade de dados armazenados. Fazíamos isso através da criação de arrays muito maiores do que o necessário, pois não sabíamos o que estaria por vir.

Com o Python não precisamos saber o tamanho real de nossa lista. Nós podemos adicionar um novo elemento, por exemplo, utilizando a função `append`, conforme demonstra o exemplo abaixo:

Exemplo `append(objeto)`

Para adicionarmos um novo item no final de nossa lista basta utilizarmos o `append`, conforme exemplo abaixo:

Anotações	

```
listaNomes = ['Gustavo', 'Pedro']
listaNomes.append('Geremias')
print(listaNomes[2])
```

operacoes X

C:\Users\Acer\PycharmProjects\Hello

Geremias

Process finished with exit code 0

Removendo um objeto do nosso Array

Para removermos o último item da nossa lista temos o comando pop, conforme demonstra abaixo:

	Anotações

```
listaNomes = ['Gustavo', 'Pedro']
listaNomes.pop()
print(listaNomes)
```

operacoes x

C:\Users\Acer\PycharmProjects\HelloWorld

['Gustavo']

Process finished with exit code 0

Del é o comando utilizado para remover uma posição específica de um vetor na nossa lista, conforme demonstra a imagem abaixo:

Anotações	

```
listaNomes = ['Gustavo', 'Pedro']
del(listaNomes[0])
print(listaNomes)
```

operacoes x

C:\Users\Acer\PycharmProjects\HelloWorld\

['Pedro']

Process finished with exit code 0

Variáveis indexadas multidimensionais (Matrizes)

Entendemos como funcionam os arrays/vetores em qualquer linguagem de programação específica como o python.

Outra possibilidade de armazenamento de informações indexadas em variáveis é a matriz. A matriz é uma espécie de Arrays de Arrays.

Podemos armazenar estruturas indexadas com mais de uma dimensão, conforme os exemplos abaixo:

	Anotações

```
arrayBidimensional = [[1, 2], [3, 4], [5, 6], [7, 8]]
linhas = 0
colunas = 0
for item in arrayBidimensional:
    linhas += 1
    colunas = 0
    for val in item:
        colunas += 1
    print(f'linha {linhas} coluna {colunas} valor {val}')
```

for item in arrayBidimensional for val in item

operacoes x

C:\Users\Acer\PycharmProjects\HelloWorld\venv\Scripts\python.exe

```
linha 1 coluna 1 valor 1
linha 1 coluna 2 valor 2
linha 2 coluna 1 valor 3
linha 2 coluna 2 valor 4
linha 3 coluna 1 valor 5
linha 3 coluna 2 valor 6
linha 4 coluna 1 valor 7
linha 4 coluna 2 valor 8
```

Dessa maneira declaramos uma matriz de inteiros que terá duas dimensões, sendo um conjunto de arrays dentro de outro array, com valores de linhas por colunas, como foi mostrado no exemplo acima. Essa matriz é uma matriz de 4 linhas por 2 colunas.

Anotações	

Exercícios

1 – Complete:

- Quando sabemos exatamente o tamanho que terá nossa coleção utilizamos _____. E quando não sabemos ao certo quantos valores existirão em nossa coleção utilizamos _____.
- _____ ou _____ são conjuntos de variáveis do mesmo tipo.
- _____ são conjuntos de arrays também chamados de _____

2 – Crie um vetor com 10 posições do tipo inteiro, preencha seus valores “hard coded”. Para este vetor crie dinamicamente uma lista do tipo inteiro que receba a multiplicação de cada valor do vetor por dez. Ao final exiba todos os valores de ambos na tela.

3 – Crie uma matriz que receba o nome de 5 pessoas. Posterior ao preenchimento dos primeiros nomes, receba o segundo nome dessas mesmas pessoas, dizendo “Qual é sobrenome do”, concatenado o primeiro nome. No final exiba todos os nomes e sobrenomes na tela.

	Anotações

Capítulo 14 – Python - Programação Orientada a Objetos

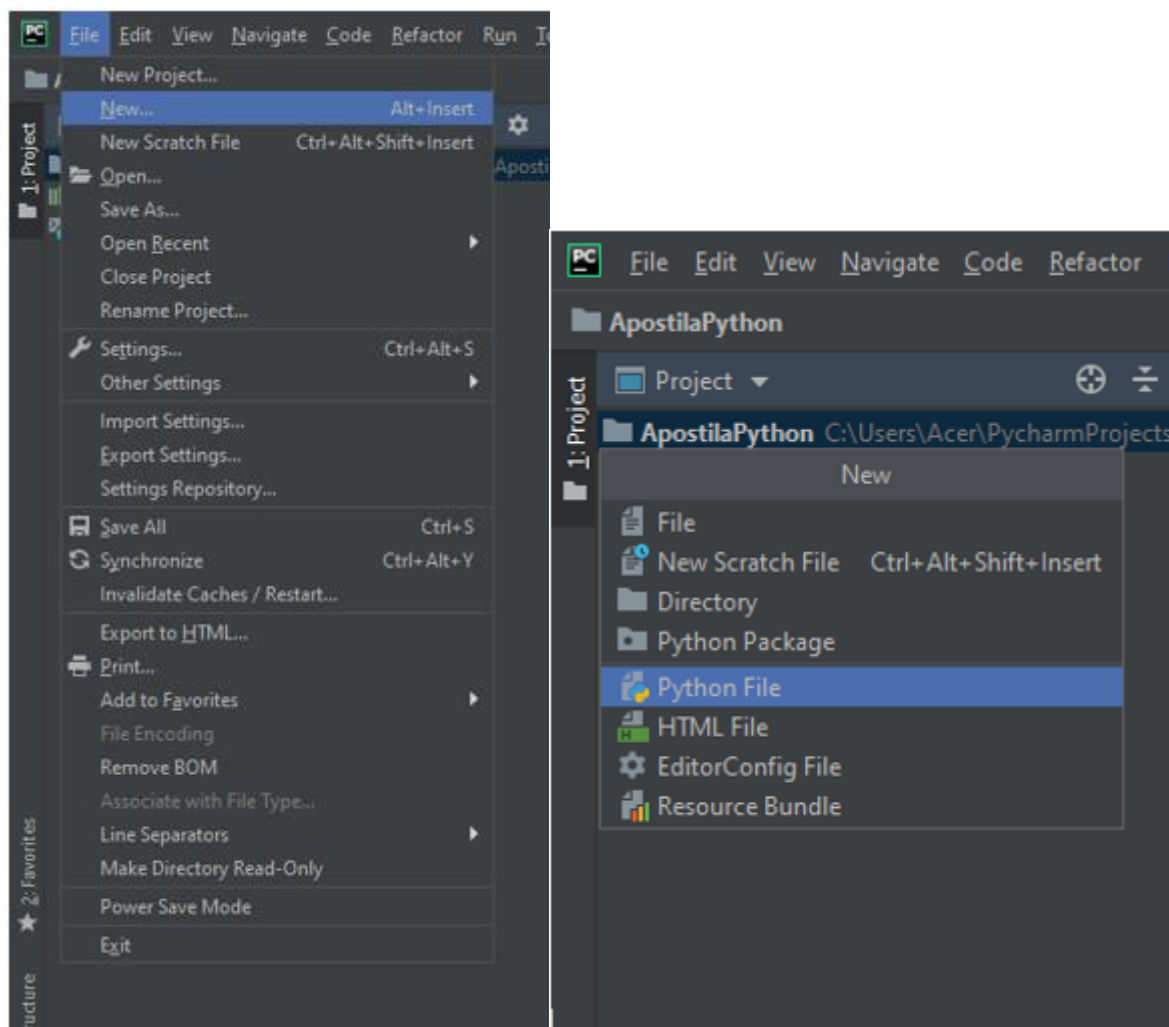
Até nesse momento do nosso curso, utilizamos as estruturas básicas da linguagem python.

Agora entraremos nas principais funcionalidades que diferem o python de outras linguagens tradicionais: a orientação a objetos.

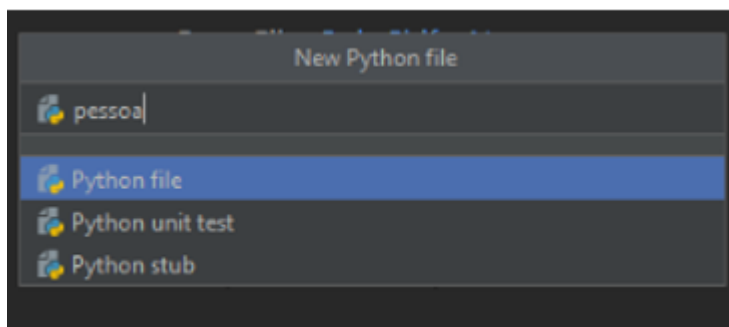
Devemos lembrar, antes de mais nada, que conceitos como classes, objetos, herança e mensagem já devem ter sido entendidos previamente quando vocês aprenderam a base da Lógica de Programação.

Caso você não lembre, temos um Curso Online disponível na Área do Aluno, onde você pode relembrar este conteúdo.

Para esse modulo ficar mais didático, vamos criar um projeto orientado a objetos. Para isso, siga os comandos mostrados no exemplo abaixo:



Anotações	



Após termos criado o nosso arquivo python, vamos agora criar os nossos parâmetros para a nossa pessoa, como por exemplo nome, endereço, idade e diaNascimento.

	Anotações

Classes

```
class Pessoa(object):
    def setNome(self, nome):
        self.nome = nome

    def setEndereco(self, endereco):
        self.endereco = endereco

    def setIdade(self, idade):
        self.idade = idade

    def setDiaNascimento(self, diaNascimento):
        self.diaNascimento = diaNascimento

    def getNome(self):
        return self.nome

    def getEndereco(self):
        return self.endereco

    def getIdade(self):
        return self.idade

    def getDiaNascimento(self):
        return self.diaNascimento
```

Preste atenção nos detalhes:

- Class é a nossa classe e criamos métodos set para setar valores em nossa classe e depois criamos métodos get para retornar os valores da classe.
- Self significa que são itens que pertencem a nossa classe, por exemplo self.nome significa que é o nome que pertence a nossa classe.

Anotações	

```
def setIdade(self, idade):
    self.idade = idade

def setDiaNascimento(self, diaNascimento):
    self.diaNascimento = diaNascimento

def getNome(self):
    return self.nome

def getEndereco(self):
    return self.endereco

def getIdade(self):
    return self.idade

def getDiaNascimento(self):
    return self.diaNascimento
```

Depois de termos criado a nossa classe, agora vamos chamar ela em outro arquivo no qual vamos fazer o cadastro da nossa pessoa, conforme demonstra o exemplo abaixo:

```
import pessoa

p = pessoa.Pessoa()
p.setNome(' Gustavo Rosauro ')
p.setEndereco(' Rua lauro muller 370 ')
p.setIdade(27)
p.setDiaNascimento('23/02/1992')

print(f' o nome da pessoa é {p.getNome()} a idade é {p.getIdade()} o endereco é {p.getEndereco()} '
      f' e a data de nascimento é {p.getDiaNascimento()}')
```

Para podermos preencher a nossa pessoa temos de importar o nosso arquivo para a página e depois chamar a nossa pessoa como no exemplo, utilizando `pessoa.Pessoa()` 'nome do arquivo.nome da classe'.

Conforme demonstra o projeto, os sets que criamos é aonde definimos os valores de nossa classe, e os gets são aonde buscamos esses valores.

	Anotações

Nesse exemplo a pessoa podia ou não informar os valores, mas e se esses valores fossem obrigatórios para nossa classe como faríamos?

Nesse caso teríamos que criar um construtor conforme, demonstra o exemplo abaixo:

```
class Pessoa(object):
    def __init__(self, nome, endereco, idade, diaNascimento):
        self.nome = nome
        self.endereco = endereco
        self.idade = idade
        self.diaNascimento = diaNascimento
    # def setNome(self, nome):
    #     self.nome = nome
    #
```

O def __init__ seria o nosso construtor e a partir de agora para que a nossa classe seja acessada temos que passar os parâmetros assim que instanciamos nossa classe.

```
import pessoa

p = pessoa.Pessoa('Gustavo Rosauero', 'Rua Lauro Muller 370', 27, '23/02/1992')

print(f' o nome da pessoa é {p.nome} a idade é {p.idade} o endereco é {p.endereco} '
      f' e a data de nascimento é {p.diaNascimento}')
```

Anotações	

Herança

Herança é o processo de especialização de classes. Em Python podemos especializar uma classe utilizando import e adicionando ela dentro do parametro da classe, conforme exemplo abaixo.

Exemplo:

```
import pessoa

class Aluno(pessoa.Pessoa):
    def __init__(self, nome, endereco, idade, diaNascimento, numeroMatricula, curso, escola):
        super().__init__(nome, endereco, idade, diaNascimento)
        self.numeroMatricula = numeroMatricula
        self.curso = curso
        self.escola = escola
```

O método super é o nosso construtor da classe mãe, que é a nossa classe principal da qual a classe aluno herda. Então como ele precisa do nome endereço e idade para ser acessada passamos esses valores pelos super e nesse mesmo momento a nossa classe aluno herda todos os atributos da classe Pessoa.

```
import pessoa
import aluno

a = aluno.Aluno('Gustavo Rosauro', 'Rua Lauro Muller 370', 27, '23/02/1992', '23423442', 'Ce', 'Apex')
print(a.nome)
print(f' o nome da pessoa é {a.nome} a idade é {a.idade} o endereço é {a.endereco} '
      f' e a data de nascimento é {a.diaNascimento} matricula {a.numeroMatricula} curso {a.curso} escola {a.escola}')
```

Polimorfismo

Polimorfismo está totalmente relacionado ao processo de Herança. É o princípio pelo qual duas ou mais classes derivadas de uma mesma superclasse podem invocar métodos que têm a mesma identificação (assinatura), mas comportamentos distintos especializados para cada classe derivada, usando para uma referência a um objeto do tipo da superclasse.

Para exemplificar esse conceito, criaremos uma pasta chamada polimorfismo a três classes, Veiculo, Automóvel e Barco, sendo automóvel e barco especializações de veículo. A estrutura das classes ficará dessa maneira:

	Anotações

Veiculo

```
class Veiculo(object):
    def __init__(self, tipoModelo):
        self.tipoModelo = tipoModelo
    def mover(self):
        print('movendo veiculo')
    def parar(self):
        print('parando veiculo')
```

Barco

```
import veiculo
class Barco(veiculo.Veiculo):
    def __init__(self, tipoModelo):
        super().__init__(tipoModelo)
    def mover(self):
        print('movendo barco')
    def parar(self):
        print('parando barco')
```

Anotações	

Automóvel

```
import veiculo

class Automovel(veiculo.Veiculo):
    def __init__(self, tipoModelo):
        super().__init__(tipoModelo)

    def mover(self):
        print('movendo o automovel')

    def parar(self):
        print('parando o automovel')
```

	Anotações

Main

```
import veiculo
import automovel
import barco

Veiculo = veiculo.Veiculo
Automovel = automovel.Automovel
Barco = barco.Barco

veiculo = []
veiculo.append(Automovel('BMW'))
veiculo.append(Barco('Phanthon'))

def movimentarVeiculo(v):
    print(v.tipoModelo)
    v.mover()

def pararVeiculo(v):
    print(v.tipoModelo)
    v.parar()

for v in veiculo:
    movimentarVeiculo(v)
    input('Digite enter para parar o veiculo')
    pararVeiculo(v)
```

Entendendo o que fizemos

A nossa classe veículo é como se fosse uma classe abstrata no qual todas as outras herdam dela, que contém o nosso parâmetro tipo modelo e as assinaturas dos nossos métodos no qual cada classe subscreve com seu próprio retorno (chamamos isso de sobrecarga) .

As classes abstratas vieram para nos fornecer um padrão de desenvolvimento do sistema.

Anotações	

Utilizando o nosso exemplo de polimorfismo podemos entender que nós nunca teremos um Objeto do tipo Veiculo puramente, pois ele sempre será Automovel ou Barco.

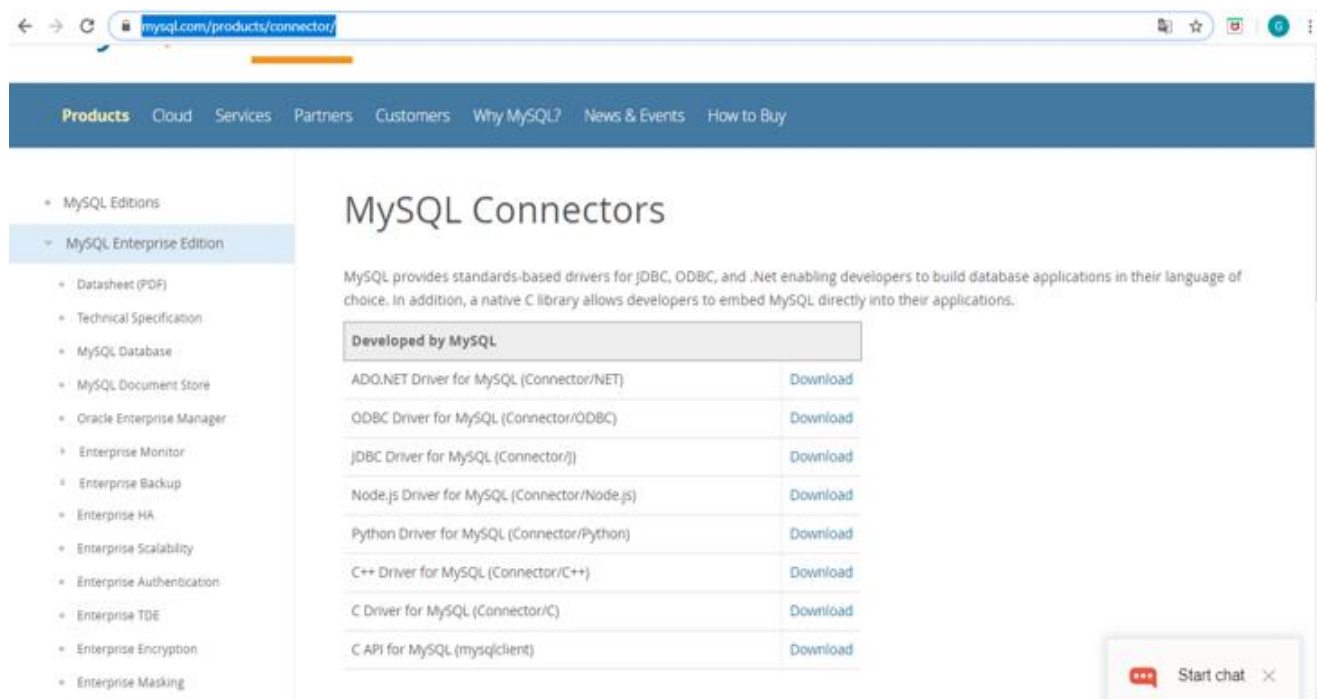
Exercícios

- 1) Crie um projeto e nomeie-o de “PythonPOO”. Seu projeto deverá:
 - a. Conter classes de animais. (Animais sugeridos, Pato, Gato, Cachorro)
 - b. Cada animal deve obrigatoriamente herdar de uma superclasse, sendo essa abstrata.
 - c. A superclasse deve conter o método abstrato EmitirSom ().
 - d. Cada um dos animais deve ser capaz de emitir som de seu próprio jeito.
 - e. O método principal deve ser capaz de visualizar as propriedades dos objetos, porém somente através do método CadastrarAnimal () que as subclasses sobrescreverão (como sobrecarga) da classe pai, será possível cadastra-los.
 - f. O método principal deve cadastrar cada um dos animais criados e atribui-los a um array da classe pai.
 - g. O método principal deve emitir o som de cada animal.

	Anotações

Capítulo 15 – Conectando ao banco de dados Mysql

Para iniciarmos este capítulo, precisamos acessar o site <https://www.mysql.com/products/connector/> para que possamos instalar o nosso conector do mysql para python.



Após instalado o conector agora vamos criar a nossa aplicação e vamos chamar de BancoMysql.

Após termos criado o nosso projeto, temos de instalar a biblioteca do mysql em nossa aplicação com o seguinte comando.

```
Terminal: Local x +
Microsoft Windows [versão 10.0.18362.535]
(c) 2019 Microsoft Corporation. Todos os direitos reservados.

(venv) C:\Users\Acer\PycharmProjects\BancoMysql>pip install mysql-connector-python
```

Anotações	

Após termos executado esse comando podemos conectar ao nosso banco de dados mysql.

Agora podemos importar o nosso conector do mysql com o seguinte comando import: mysql.connector.

Após termos importado o nosso conector, agora estabeleceremos a conexão passando as credenciais do nosso banco de dados, conforme exemplo abaixo:

```
import mysql.connector

db = mysql.connector.connect(
    host='localhost',
    database='apostila',
    user='root',
    password=''
)

try:
    cursor = db.cursor()
    print('Conectado ao banco de dados com sucesso!')
    cursor.close()
except print(Exception):
    pass
```

Host é nosso servidor

Database é o nome da nossa base de dados

User é o nosso usuário

Password é a senha

Estabelecida a conexão com o nosso banco de dados, agora podemos inserir, consultar, editar, e remover dados da nossa base utilizando o python.

	Anotações

Inserindo um dado em nossa base

Agora que estabelecemos a conexão com nossa base de dados vamos criar uma tabela para podermos inserir nossos dados. O nome dessa tabela pode ser pessoa e terá ID, NOME e IDADE.

```
import mysql.connector

db = mysql.connector.connect(
    host='localhost',
    database='apostila',
    user='root',
    password=''
)

try:
    cursor = db.cursor()
    sql = 'INSERT INTO PESSOA (NOME, IDADE) VALUES (%s, %s)'
    records = ('Gustavo Rosauco', 27)
    cursor.execute(sql, records)
    db.commit()
    cursor.close()
    print('nova linha inserida')
except Exception as e:
    pass
```

Nessa imagem podemos observar que por meio do nosso cursor executamos o comando sql junto com nossos records, substituímos os charetesres % pelos valores do nosso records e quando executamos podemos consultar o no banco e veremos que o registro foi inserido.

Anotações	

Consultando dados na base

Para fazermos a consulta agora iremos utilizar o comando select da seguinte maneira:

```
import mysql.connector

db = mysql.connector.connect(
    host='localhost',
    database='apostila',
    user='root',
    password=''
)

try:
    cursor = db.cursor()
    sql = 'SELECT * FROM PESSOA'
    cursor.execute(sql)
    resultado = cursor.fetchall()
    for row in resultado:
        print(f'nome {row[1]} e idade {row[2]}')
    cursor.close()
except Exception:
    pass
```

Nosso cursor executa o comando e logo em seguida utilizamos o comando `fetchall()` que serve para capturarmos os dados de retorno da nossa consulta.

Com esses dados em memória utilizamos um laço repetição para ler linha a linha o nosso retorno, depois acessamos os valores da coluna 1 e da coluna 2.

A coluna 0 é o nosso id e nesse caso não nos interessa

	Anotações

Alterar um registro

Para alterarmos um registro utilizaremos o comando update conforme o exemplo abaixo:

```
password=' '
)
id = 1
try:
    cursor = db.cursor()
    sql = 'UPDATE PESSOA SET NOME = %s WHERE ID = %s'
    params = ('Gustavo Rosauo Professor', id)
    cursor.execute(sql, params)
    db.commit()
    cursor.close()
except Exception:
    pass
```

Anotações	

Remover um registro

```
import mysql.connector

db = mysql.connector.connect(
    host='localhost',
    database='apostila',
    user='root',
    password=''
)
id = 1
try:
    cursor = db.cursor()
    sql = f'DELETE FROM PESSOA WHERE ID = {id}'
    cursor.execute(sql)
    db.commit()
    cursor.close()
except print(Exception):
    pass
```

	Anotações

Exercícios

1 - Crie um projeto chamado cadastro e dentro desse projeto faça uma lógica para:

- Salvar um novo usuário passando nome, e-mail, cargo e idade
- Quando quiser alterar um usuário já cadastrado, tem de ter uma opção para que eu possa ver todos os usuários já cadastrados e possa escolher o número de um para alterar
- Tem de ter uma opção para excluir usuários, também antes mostrando todos os usuários já cadastrados e escolhendo um número para ser removido

Obs: Quando iniciar a aplicação temos de perguntar a o usuário qual operação ele deseja realizar, sempre seguindo os tópicos acima.

Para esse exercício utilizaremos o banco de dados e a orientação a objetos.

2 – No mesmo Exercício crie um filtro de dados para poder pesquisar um usuário pelo nome

Anotações	

Capítulo 16 – Utilizando Filter no Python

O filter serve para filtrar listas, utilizando uma expressão lambda conforme exemplo abaixo:

```
listaNomes = ['Gustavo', 'Pedro', 'Maria', 'João']

novaLista = list(filter(lambda x: x == 'Gustavo', listaNomes))

print(novaLista)
```

Aonde o lambda irá chamar cada item de nossa lista como x e depois irá fazer uma comparação e consultar a lista retornando o que estiver dentro de nossa condição. No caso do exemplo acima retornando somente aonde encontrar 'Gustavo'.

Caso estivermos trabalhando com objetos em um formato json, podemos filtrar da seguinte maneira:

```
listaPessoas = [
    {
        "nome": 'Gustavo',
        "idade": 27
    },
    {
        "nome": 'Ralph',
        "idade": 24
    },
    {
        "nome": 'Jefferson',
        "idade": 15
    }
]

listaPessoas = list(filter(lambda x: x["idade"] > 18, listaPessoas))
print(listaPessoas)
```

Bastando somente acessar o parâmetro daquele objeto e a nossa condição.

Quando formos trabalhar com um objeto nosso podemos utilizar da seguinte maneira:

	Anotações

```
class Pessoa:
    def __init__(self, nome, idade):
        self.nome = nome
        self.idade = idade

listaPessoas = []
p1 = Pessoa('Gustavo', 27)
listaPessoas.append(p1)
p2 = Pessoa('Lucas', 15)
listaPessoas.append(p2)
listaPessoas = list(filter(lambda x: x.nome == 'Gustavo', listaPessoas))
for p in listaPessoas:
    print(p.nome)
```

	Anotações