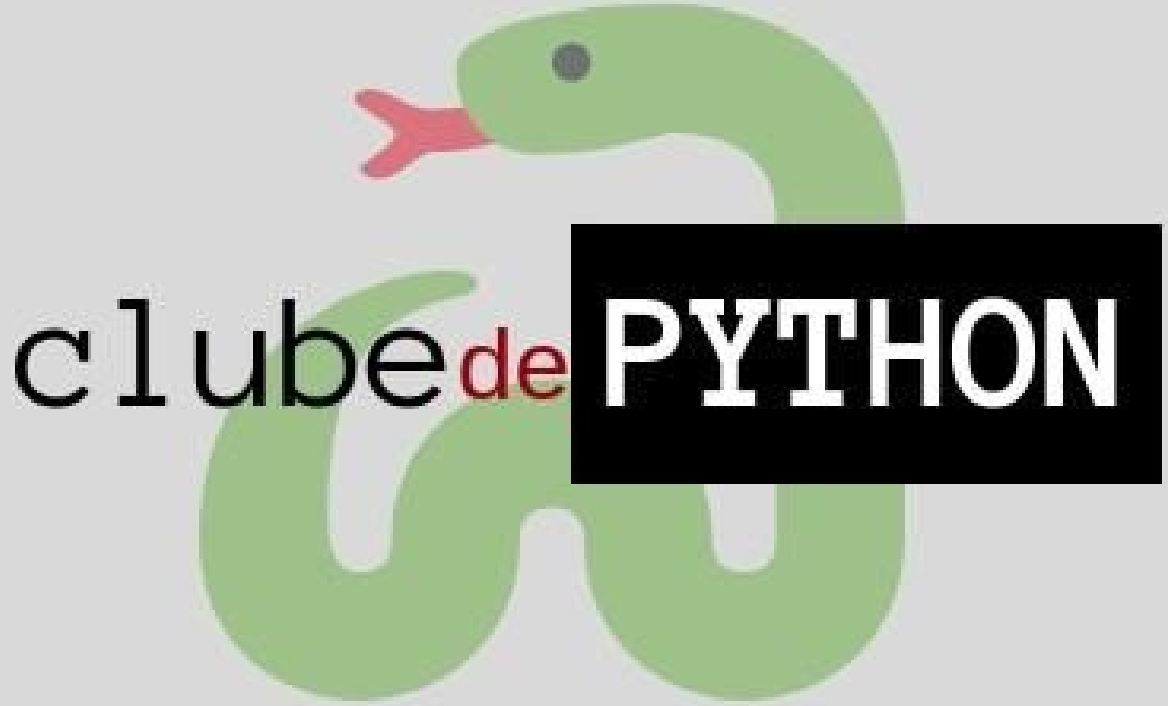
clubedePYTHON

Programa Interagir

O programa “Interagir: Integração entre Ciência, Computação e Sociedade” possui eixos principais relacionados: o primeiro visa disseminar a Ciência e a Computação, e incentivar profissionais, professores e estudantes das áreas relacionadas (e.g. computação, matemática, engenharias, tecnologias, educação, etc.) a aprofundarem seus conhecimentos e habilidades. O segundo visa promover o nome da UDESC, enquanto instituição de ensino, pesquisa e extensão de qualidade, apta a contribuir com o desenvolvimento social, científico e tecnológico. Os eixos são apoiados por 3 projetos que englobam diversas ações.

INTERAGIR

Eu quero programar	Catarinas	Cientistas Anônimos
Ensino de Algoritmos e Programação para a comunidade, que de um lado faz a conexão entre ensino-extensão, onde estudantes estarão envolvidos na discussão da temática, visando a melhoria dos ambientes computacionais com a sociedade por meio de desenvolvimento tecnológico; e que de outro lado, compreende as questões técnicas para a realização de minicursos e planejamento de materiais para oferta destes à comunidade.	Trata de ações afirmativas para envolvimento das mulheres na ciência, fortalecer e incentivar a presença da mulher na tecnologia por meio de oficinas, encontros e debates, abertos a comunidade sempre promovendo a igualdade e a inclusão. Catarinas	Visa auxiliar estudantes a fazer ciência, apoiar a apresentação a produção científica e internacionalização da universidade. Faz a conexão da pesquisa-extensão de maneira que estudantes possam aprimorar suas técnicas de apresentação e comunicação, fazendo a difusão informacional da UDESC na comunidade. Trata de ações sociais e de divulgação com a comunidade local e regional.

Comandos básicos - Linux

Manipulação de arquivos e pastas	
Comando	**Descrição**
$ ls	Exibe arquivos do diretório
$ cd $ cd ..	Modifica o diretório atual Modifica para diretório pai / anterior
$ mkdir nome_pasta	Cria uma nova pasta no diretório atual
$ cp arq_origem pasta_destino/ $ cp -r pasta_origem dir_destino	Copia arquivo para pasta destino Copia recursivamente a pasta origem para o diretório destino
$ mv $ mv arq_origem pasta_destino/	Move arquivos e pastas Move o arquivo origem para a pasta destino
$ rm $ rm –rf pasta	Remove arquivos e pastas Remove recursivamente os arquivos da pasta
$ cat arquivo	Exibe o conteúdo do arquivo
$ diff arq1 arq2	Exibe as diferenças entre dois arquivos
$ find $ find /pasta/desejada nome_arq	Busca arquivos e diretórios a partir de um diretório dado Busca por ‘nome_arq’ no diretório ‘/pasta/desejada’
Manipulação do terminal/prompt	
Comando	**Descrição**
$ man comando_a	Exibe o manual do comando_a; Pressionar q para sai
$ su	Permite alternar entre usuários do sistema. “su” alterna para o usuário administrador (root)
$ history	Exibe os últimos comandos executados pelo usuário
$ top	Exibe informações sobre os processos e uso de recursos do computador
$ pwd	Exibe o caminho do diretório atual
$ clear	Limpa todo o conteúdo do terminal
$ exit	Fechar janela do terminal
Ctrl+C	Para processo em andameto
Ctrl+Alt+T	Abre uma nova guia no terminal no diretório raiz
Ctrl+Shift+T	Abre uma nova aba no terminal no diretório atual
Ctrl+Shift+W	Fecha a aba do terminal
Ctrl+PageUp	Altera visualização para aba/terminal anterior
Ctrl+PageDown	Altera visualização para próxima aba/terminal

Introdução ao Python

Criada no final dos anos oitenta por Guido van Rossum no Centro de Matemática e Tecnologia da Informação (CWN) na Holanda, para ser sucessora da linguagem de programação ABC. O Nome vem do gosto do criador pelos humoristas britânicos Monty Python.
Python é uma linguagem de programação de alto nível, interpretada, com tipagem dinâmica e forte, além de ser multiparadigma (orientado a objetos, imperativo, funcional e processual), criada para ser simples e possuir um código mais legível.
Atualmente é gerenciada pela Python Software Foundation e possui licença de código aberto e sua versão mais recente é a 3.7.

Por que Python?

- Simplicidade: Comparada a outras linguagens podemos ver códigos em python são menores, mais objetivos e legíveis.
- Multiplataforma: Programas escritos em uma plataforma serão executados na maioria das outras plataformas sem problemas. Caso a plataforma não possua uma versão de python é possível modificar o código da linguagem para que ela rode onde for necessário.
- Robustez: Os programas em python são distribuídos na forma de código-fonte, dessa forma qualquer pessoa pode alterar, corrigir e melhorar os algoritmos. Isso torna os módulos mais seguros e estes são testados diversas vezes sob diversos aspectos, alcançando maior robustez.
- Comunidade: Tem uma comunidade ativa espalhada pelo mundo todo, e por ser uma linguagem livre, todos podem contribuir fazendo com que haja sempre muito material disponível.

Python é uma linguagem simples e expressiva, ou seja, com poucas linhas de código se faz muito. Além disso permite, através de diversos módulos e frameworks, ser aplicada em quase todas as áreas da computação como desenvolvimento web, banco de dados, desenvolvimento de softwares e jogos, computação gráfica, processamento de imagem, computação distribuída, entre outros.

Por onde começar?

Tutorial de como instalar no ubuntu e no windows

```
$ python3 --version                 # verificar se já está instalado
$ sudo apt-get install python3      # instalar python 3
$ atom arquivo.py                   # criar arquivo python no editor atom
$ subl arquivo.py                   # criar arquivo python no editor sublime
```

Executar programas pelo terminal: helloword.py
Módulos/bibliotecas: time, random, math, pygame.

Variáveis e tipos

O python automaticamente define o tipo da variável e o seu local da memória. Então as variáveis não precisam ser declaradas (pelo seu tipo), mas elas precisam ser inicializadas antes de serem utilizadas. Para inicializar as variáveis usamos o operador de atribuição =.
Podemos mudar o tipo de uma variável ao atribuir um valor de outro tipo à ela.

Ex: x = 1 (x é int)
x = ‘1’ (x agora é string)

Para consultarmos o tipo da variável utilizamos a função type(variável).

Tipos essenciais:

int	>>> quantidade = 1
	>>> type(a)
	<type 'int'>

float	>>> peso = 1.0
	>>> type(a)
	<type 'float'>

bool	>>> condicao = True
	>>> type(a)
	<type 'bool'>

Operadores e comparações

Operadores Aritméticos		Operadores de comparação		Operadores lógicos	
+	Adição	==	Igual (valores)	and	E
-	Subtração	>	Maior	or	Ou
*	Multiplicação	<	Menor	not	Negação
/	Divisão	>=	Maior ou igual		
%	Resto da divisão	<=	Menor ou igual		
**	Potenciação	!=	Diferente		

Loops

Python reconhece o espaço/tab como as ações dentro dos operadores condicionais. Por isso, é preciso sempre respeitar a formatação da linha utilizando tab após o if, elif ou else.

Ao invés de utilizar chaves para denotar esses blocos, após o operador é utilizado dois pontos.

```
if (primeira_condição):
    ...
elif (segunda_condição)
    …
else:
    ....
```

Fluxo condicional

O **if** é uma estrutura de condição, que ao avaliar uma expressão permite executar ou não uma determinada ação. A estrutura é dada pela palavra reservada **if**, seguida pela condição e por dois pontos.

Para definirmos um comportamento específico para quando a condição não for válida, utilizamos a palavra reservada **else**.

Se existir mais de uma condição alternativa que precisa ser verificada, utilizamos o **elif** para avaliar as expressões.

Em linguagens de baixo nível como C, utilizamos
if(){
}else if(){
}else{
}
que em python foi abreviado para elif, reduzindo código.

Loops também chamados de estruturas de repetição, são blocos de código que são executados repetidas vezes enquanto uma determinada condição é satisfeita.

For: O laço **for** permite percorrer um conjunto de dados, executando para cada um deles uma determinada ação.
A estrutura de um laço **for** é a seguinte:
for variavel **in** lista
    comandos

Ao percorrer a lista, a variável indicada no **for** recebe a iteração.

While: A palavra reservada **while** faz com que uma instrução seja executada enquanto uma determinada condição é atendida.
A estrutura é dada pela palavra reservada **while** seguida pela condição entre parênteses e dois pontos, e logo abaixo, indentadas, as instruções que devem ser executadas a cada iteração.

Tipos complexos:Arrays,Strings, Lists, Tuplas, Dictionarios.

Strings: Todas as strings são do tipo str.

```
>>> a = ‘Python’
>>> type(a)
<type 'str'>
```

Na atribuição de um valor do tipo string, não há diferenciação entre aspas simples ou duplas.

Uma string pode ser iterável:

```
>>> for letra in 'Python':
...     print letra
...
P
y
t
h
o
n
```

Podemos acessar os valores da string pelo seu índice (mas não podemos alterar os valores dessa maneira).

```
>>> a = 'Python'
>>> a[0]
'P'
```

Para descobrirmos o tamanho de uma string utilizamos a função len().

```
>>> a = 'Python'
>>> len(a)
6
```

Lista: É um conjunto ordenado de valores onde cada valor pode ser acessado a partir do seu índice. Para criar uma lista colocamos os valores entre colchetes. Os elementos de uma lista podem ser de tipos diferentes.

A função len() nos retorna o número de elementos da lista.Para verificarmos se um elemento faz parte da lista, utilizamos o operador in.

```
>>> list = [1, 2.3,
"joao"]
>>> list[1]
2.3
```

```
>>> list=[1, 2, 3, 4]
>>> len(list)
4
```

```
>>> list=[10, 20, 30,
40]
>>> 30 in list
True
>>> 50 in list
False
```

Para adicionarmos um elemento no final da lista utilizamos a função append(), para adicionarmos um elemento em alguma posição utilizamos a função insert() e para excluir um elemento utilizamos a função remove().

```
>>> list = [1, 2, 3, 4, 5]
>>> list.append(6)
>>> print list
[1, 2, 3, 4, 5, 6]
```

```
>>> list = [1, 2, 3, 4, 5]
>>> list.insert(1, 7)
>>> print list
[1, 7, 2, 3, 4, 5]
```

```
>>> list = [1, 2, 3, 4,
5]
>>> list.remove(2)
>>> print list
[1, 3, 4, 5]
```

Comparando listas: Listas são comparadas lexicograficamente. Se duas listas são iguais até os k- é-simos elementos, o resultado da comparação depende da comparação entre os (k+1) ésimos elementos Se alguma das listas tem somente k elementos, então esta é a menor. Duas listas são iguais se e somente se têm o mesmo comprimento e todos os elementos de mesma posição são iguais. Uma lista é maior que um número mas menor que uma string.

>>> [1,2] < [2, 3]
True
>>> [1,2] < [1, 2, 3]
True
>>> [1,2] != [1,2]
False
>>> min([[1],[2,3],[3,4],[]])
[]
>>> max([[1],[2,3],[3,4],[]])
[3, 4]
>>> min(0,[],"")
0
>>> max(0,[],"")
"
	lista.reverse()
	>>> lista
	[3, 2, 1]
		STRING + LISTA
		
		s = "hello, world!"
		>>> print s[0:5]
		//pega do 0 ate 4 caracter
		'hello'
		>>> print s[:5]
		'hello'
		>>> print s[2:4]
		ll
		>>> print s[7:13]
		'world!'
		>>> print s[7:]
		'world!'
		>>> print s[:]
		'hello, world!'

Tuplas: É um conjunto imutável de valores. Utilizamos isso para definir uma conjunto de valores que não podem ser mudados em todo o programa. Por exemplo, podemos fazer uma tupla com os dias da semana pois eles sempre são os mesmos. Para inicializar uma tupla, colocamos seus valores separados por vírgulas.

```
>>> tupla = ‘Domingo’, ‘Segunda’, ‘Terca’, ‘Quarta’, ‘Quinta’, ‘Sexta’,
‘Sabado’
>>> type(tupla)
<type ‘tuple’>
>>> tupla[0]
‘Domingo’
```

Dicionários: Lista de itens e chaves. Seus elementos são conjuntos de pares (chave - valor).
Para criamos um dicionário, inicializamos seus elementos entre chaves.
As chaves e valores podem ser de qualquer tipo.

Por exemplo, podemos ter chaves strings e valores inteiros.

```
>>> dicionario = {"chave1": 1, "chave2": 2, "chave3": 3}
>>> print(dicionario)
{'chave1': 1, 'chave2': 2, 'chave3': 3}
```

Para buscarmos algum valor, só precisamos chamar pela sua chave referente.

```
>>> dicionario = {"banana":1, "maca":2, "laranja":3, "uva":4}
>>> print(dicionario["laranja"])
3
```

Podemos utilizar o método has_key() para verificar se a chave existe dentro do dicionário e a função del() para deletarmos algum elemento a partir da sua chave.

```
>>> dicionario.has_key(“banana”)
True
```

```
>>> dicionario = {"banana":1, "maca":2}
>>> del dicionario[“banana”]
>>> print(dicionario)
{'maca': 2}
```

Funções

Uma função é uma sequência de comandos que executa alguma tarefa e tem um nome.
Os nomes de funções podem ser qualquer um, porém não podem ser palavras reservadas do Python.

Formato geral:

```
def nome_da_funcao (arg1, arg2, …, argn):
        comando
        ...
        comando
        return expressao
```

Onde:

nome_da_funcao: nome da função, ao qual deverá ser chamada no fluxo principal do programa.

argn: argumentos da função, podem ser de tipos diferentes, e entre 0 e n argumentos;

comando: os comandos que serão executados dentro da função

return expressao: resultado retornado a quem chamou a função, pode ser de qualquer tipo ou nulo como ‘return null’

Chamadas de função são um desvio no fluxo de execução. Em vez de ir pro próximo comando, o fluxo salta para a primeira linha da função chamada, executa todos os comandos lá e depois volta atrás para retomar de onde havia parado no fluxo principal.

Exemplos:

```
def f():
   return
```

```
$ print f()
None
```

```
def f():
```

```
$ print f()
```

return ‘Oi’	Oi
def f(name): return ‘Oi ’+name	$ print f(‘Bruna’) Oi Bruna

É possível criar e utilizar variáveis dentro de funções, porém é importante lembrar que elas pertencem apenas a função onde foram criadas.

Exemplo:

def imprimeN(nome, qtd):
  i=0
  for i in range(0, qtd):
  print nome
  return 0
	$ print imprimeN(‘Bruna’, 2)
	Bruna
	Bruna
	0
	$ print i
	NameError: name 'i' is not defined

Funções Lambda: Lambda são funções anônimas que aceitam argumentos (inclusive opcionais) e que só suportam uma expressão.
Ao executar lambda, Python retorna uma função ao invés de atribuí-la à um nome como acontece com def, por isso são anônimas.

Exemplo:

Usando função def	**Usando Lambda**

def size_each(words):
  return [len(w) for w in words]
 	
words = ['look', 'so', 'car', 'ice', 'melted']
 	
print size_each(words)
>>> [4, 2, 3, 3, 6]
	words = ['look', 'so', 'car', 'ice', 'melted']
	
	size_each = lambda words: [len(w) for w in words]
	
	print size_each(words)
	>>> [4, 2, 3, 3, 6]

Classes

O conceito de classes, métodos e objetos é geral para qualquer linguagem orientada a objetos. Uma classe associa dados (atributos) e operações (métodos) numa só estrutura. Um objeto é uma instância de uma classe. Ou seja, uma representação da classe.

Por exemplo, Regis é uma instância de uma classe chamada Pessoa, mas a Pessoa é a classe que o representa de uma forma genérica. Se você criar um outro objeto chamado Fabio, esse objeto também será uma instância da classe Pessoa.

```
class Pessoa:

    def __init__(self, nome):
        self.nome = nome
```

```
        def __str__(self):
                return self.nome


regis = Pessoa('Regis')
print(regis)
fabio = Pessoa('Fabio')
print(fabio)
```

Exemplo - Calculadora:
O método init é o método que inicializa os parâmetros no momento de instanciamento do objeto.

```
#calculadora.py
class Calculadora:

        def __init__(self, a, b):
                self.a = a
                self.b = b

        def soma(self):
                return self.a + self.b

        def subtrai(self):
                return self.a - self.b

        def multiplica(self):
                return self.a * self.b

        def divide(self):
                return self.a / self.b
```

```
from calculadora import Calculadora
>>> c = Calculadora(128,2)
>>> print('Soma:', c.soma())
Soma: 130
>>> print('Subtração:', c.subtrai())
Subtração: 126
>>> print('Multiplicação:', c.multiplica())
Multiplicação: 256
>>> print('Divisão:', c.divide())
Divisão: 64.0
```

Como dito antes, definimos os valores iniciais apenas uma vez e depois apenas usamos os métodos para calcular os valores.

Módulos

Módulos em python, são arquivos de código que podem ser importados por outros módulos. A modularização é importante, pois permite reutilizar o código em outras aplicações e mantém ele organizado, cumprindo com os propósitos da linguagem de ser simples e legivel.

Comunidade de Python em Joinville

A Python Joinville é uma comunidade de usuários da linguagem e do ecossistema Python de Joinville, Santa Catarina. Feito pela comunidade para a comunidade, tem o objetivo de difundir a linguagem, promover a troca de experiências e manter a comunidade crescendo igualmente em público e impacto social. Promovemos encontros onde os participantes podem contribuir para projetos de software livre e adquirir novos conhecimentos com desenvolvedores da comunidade. Sendo um hacker ou um iniciante, você será bem vindo.

Os encontros se dão por meio de meetups, que são reuniões informais onde qualquer pessoa pode levar um tema para apresentar e depois ser discutido entre os presentes. É um meio de difundir conhecimento e debater em comunidade sobre assuntos em comum.

Você pode encontrar mais informações nesse link: http://python.joinville.br/

Dentro da comunidade de python em Joinville também existe o Pyladies Joinville, que é um grupo de mulheres desenvolvedoras amantes da programação em Python. O primeiro grupo foi criado por sete mulheres em Los Angeles, Estados Unidos e logo se espalhou, tendo, atualmente, mais de 40 grupos ao redor do mundo.
Hoje existem vários grupos de pyladies espalhados pelo Brasil, além da própria Pyladies Brasil.
O objetivo é instigar mulheres a entrarem na área de tecnologia e mudar essa realidade de termos poucas garotas em uma área tão rica e ampla como a computação. O propósito não é segregar e sim criar um ambiente atrativo e com mais representatividade.

Você pode encontrar mais informações nesse link: http://brasil.pyladies.com/
Ou no site do Pyladies Joinville: pyladiesjoinville.com (Em criação)

Parabéns! Você concluiu o curso de Introdução à Python.

Venha participar do movimento: Catarinas UDESC
Entre em contato: catarinasudesc@gmail.com