python_basico

March 3, 2022

1 0.0. Introdução

Python: É uma linguagem de programação interpretada, em outras palavras precisa de um interpretador onde o código fonte é traduzido em bytecode para o computador.

Métodos Para executar programas em Python: Usando o Terminal abrindo arquivos na extensão .py

Indentação: Nada mais é um recuo, que geralmente o Python coloca para definir uma Hierarquia, por exemplo, uma Função, que representa 4 espaços.

Comentários: Utilizado para documentação no código, futuras implementações e anotações, utiliza o símbolo: # Olá Amigos ou ''' Olá Amigos '''.

2 1.0. Lógica de Programação

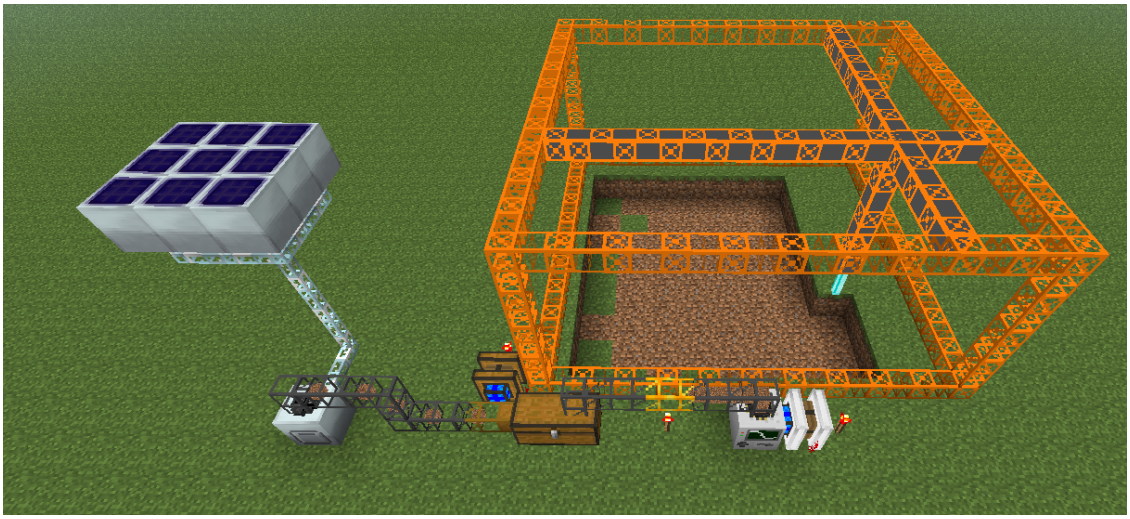
####

"A Lógica de programação nada mais é que a forma que montamos uma estrutura de passos coerentes para realizar determinada tarefa, que no caso da programação é um software para resolver algum problema."

Não é so na programação que você utiliza lógica, você utiliza desse recurso em n atividades, como por exemplo, fazer um bolo, quais são os passos para fazer um bolo ? Mas caso você gosta de jogar, vou dar o exemplo do Minecraft, vamos supor que você montou uma mineradora automática (quary) e quer também automaticamente processar os minérios que essa mineradora vai coletar, Como que você vai fazer isso?

[2]:
```
Image('../n_img/quary1.png')
```

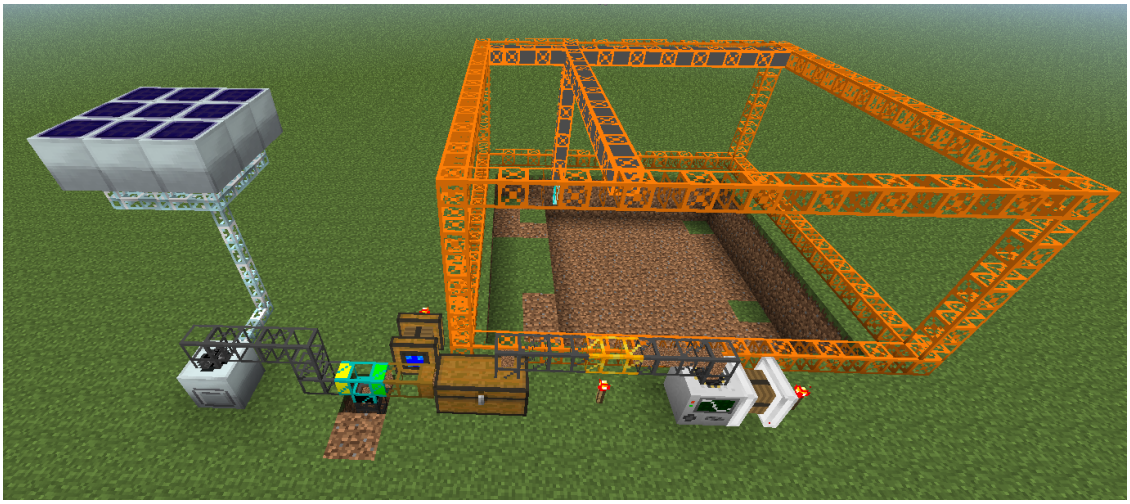
[2]:



1. Construir a Quary e desenvolver uma sequencia de tubos de transporte de itens ate um local de armazenamento desses itens. 1.1. Mas para a Quary funcionar é necessário energia, por isso o motor ao lado, se não tiver energia logo não vai ser possível coletar o minério.
2. Retirar os itens desse local de armazenamento com um motor e um tubo especifico para a retirada desse material 2.1. O motor também precisa de energia, porem é um motor mais fraquinho que so vai retirar um item por vez.
3. Movimentar os itens até a máquina que vai processar esses minérios, que no caso ali é um 'Crusher' que vai quebrar os minérios em pó. 3.1. Novamente é necessário energia, para isso foi utilizado paineis solares para ligar essa máquina.

[3]:
```
Image('../n_img/quary2.png')
```

[3]:



Porém, a máquina não sabe trabalhar com terra e materiais diferentes de minérios, então, **se não tiver terra** ela vai funcionar com os minérios, então para isso existe o cano azul, **se vim terra**

pelo cano, então transporta para outro lugar a terra, mas **se for minério**, então leva até a máquina para ser processado.

Bem, ainda temos que lidar a máquina, após ela processar os minérios, esses resultados do processamento vão para qual lugar? Outro armazenamento? alguma máquina para processar o pó do minério?

Em outras palavras, você utiliza lógica em todo momento, existem várias formas de processar minérios no Minecraft, montar um Bolo e ate mesmo construir um software.

3 1.0. Introdução ao Python

3.1 1.1. Variáveis & Tipos de Dados em Python

Os tipos de dados em pythonserve para muitas finalidades, pois se você quiser desenvovler um programa que recebe a idade de um usuário e realiza alguma transformação nela, essa idade deve ser um número certo?

Em python, para declarar uma variável, basta escolher um nome seguindo algumas regras, como o nome da variavel não pode começar com número, não pode conter espaços explicitos e tem que ser antes determinada.

Mas para que serve uma variável ?Nada mais é que a forma que você vai armazenar os dados quando você esta desenvolvendo um programa, essas variáveis são armazenadas na memória do computador.

[7]:
```
# Não pode conter espaços

ola sou uma variavel
```

```
  Input In [7]
    ola sou uma variavel
        ^
SyntaxError: invalid syntax
```

[9]:
```
# A variavel não foi antes determinada, obs, NÃO USE ACENTOS!

ola_sou_uma_variavel
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Input In [9], in <cell line: 3>()
      1 # A variavel não foi antes determinada
----> 3 ola_sou_uma_variavel

NameError: name 'ola_sou_uma_variavel' is not defined
```

[10]:
```
# Atribuindo um valor a uma variavel

ola_sou_uma_variavel = 10
```

Para armazenar dados em uma variável, utiliza-se o operador = Para mostrar o resultado dessa variável, basta utilizar a função print(nome da variavel) que ela funciona para mostar o que esta sendo armazenado na variável.

[12]:
```
print(ola_sou_uma_variavel)
```

```
10
```

[14]:
```
# Uma variavel não pode ter mais que um dado armazenado dessa forma, para␣
  ↪issoutiliza-se outros tipos de dados

ola_sou_uma_variavel = 2022

print(ola_sou_uma_variavel)
```

```
2022
```

Para mostarr o formato que esta sendo armazenado o dado em uma variável, basta utilizar outra função chamada **type**

[16]:
```
type(ola_sou_uma_variavel)
```

[16]:
```
int
```

[24]:
```
# Exemplo de Variáveis do exemplo do Minecraft

# Variável de Texto ( STRING )
minerio = 'ferro'

# Variável Numérica Inteira ( INT )
qntd_de_terra_no_bau = 370

# Variavel Numérica Decimal ( FLOAT )
litros_de_conbustivel = 12.34

print(f'O Atual minerio trafegando no cano é: | {minerio} | \n')
print('A quantidade de terra no bau atualmente é: | {} | \n '.
  ↪format(qntd_de_terra_no_bau))
print('O atual combustível disponivel no Motor é: ', litros_de_conbustivel)
```

```
O Atual minerio trafegando no cano é: | ferro |

A quantidade de terra no bau atualmente é: | 370 |

O atual combustível disponivel no Motor é:  12.34
```

Repare que no exemplo acima, foi utilizado varios tipos de “prints” diferentes vamos dizer assim, basicamente são formas de formatar uma String, pois na função print, ela requer um formato de dados que pode ser uma String.

[35]:
```
'''
Primeira Forma
O f antes da String é uma forma de formatar uma string colocando uma variável
ou algum padrão dentro de chaves "{ padrão }", esses padrões podem ser␣
 ↪encontrados
no material de referência no final do Notebook.

Segunda Forma
A segunda forma você utiliza um "Método" vamos dizer assim do tipo de dado␣
 ↪String

'''
a = 50

b = f'{a}'

print(b)
```

```
50
```

[36]:
```
# Repare que o tipo de dado é uma String

type(b)
```

[36]:
```
str
```

[43]:
```
OlaAmigos = 'Olá {} Amigos'

OlaAmigosDenovo = OlaAmigos.format('Pequenos')

OlaAmigosDenvoNovo = OlaAmigosDenovo + ', Tudo bem ?'

print(OlaAmigos)
print(OlaAmigosDenovo)
print(OlaAmigosDenvoNovo)
```

```
Olá {} Amigos
Olá Pequenos Amigos
Olá Pequenos Amigos, Tudo bem ?
```

No exemplo acima foi introduzido uma nova forma de escrever uma variavel chamada de CamelCase, onde em vez de tudo minusculo separado pelo Underscore, é Maiúscula e onde tem espaços o começo. (ola_amigos OlaAmigos), porem no Python é necessário que a variavel seja igual, pois caracteres diferentes fazem diferença.

Existe também o tipo de dado Booleano, que nada mais é que condições de Verdadeiro e Falso. Vai ser detalhado melhor a funcionalidade desse tipo de dado ao decorrer do Notebook

[83]:
```
# Igual a       0 -> Falso
# Diferente de  0 -> Verdadeiro
print( 'Valor 0: ', bool( 0 ) )
print( 'Diff de 0: ', bool( -211 ) )

# String Vazia          -> Falso
# String com Registros -> Verdadeiro
print( '\nString Vazia: ', bool( '' ) )
print( 'String Não Vazia kkkk :', bool( 'Ola Mundo' ) )

# Lista vazia     -> Falso
# Lista com Dados -> Verdadeiro
print( '\nLista Vazia: ', bool( [] ) )
print( 'Lista com Dados: ', bool( ['ouro'] ) )
```

```
Valor 0:  False
Diff de 0:  True

String Vazia:  False
String Não Vazia kkkk : True

Lista Vazia:  False
Lista com Dados:  True
```

Outro tipo de dado é as **Listas**, **Sets**, **Tuplas** e os **Dicionarios**.

[68]:
```
lista_de_minerios = ['ferro', 'ouro', 'cobre', 'prata', 'chumbo', 'terra']

set_de_minerios = {'ferro', 'ouro', 'cobre', 'prata', 'chumbo', 'terra',␣
 ↪'ferro', 'ferro', 'ferro_denovo'}

tupla_de_minerios = ('ferro', 'ouro', 'cobre', 'prata', 'chumbo', 'terra')

dicionario_de_minerios = {'minerio_1': 'ferro', 'minerio_2': 'ouro', 'lixo_1':␣
 ↪'terra'}
```

[64]:
```
lista_de_minerios # O print já é automaticamente embutido no Jupyter !!
```

[64]:
```
['ferro', 'ouro', 'cobre', 'prata', 'chumbo', 'terra']
```

[67]:
```
set_de_minerios # O set não permite dados duplicados
```

[67]:
```
{'chumbo', 'cobre', 'ferro', 'ferro_denovo', 'ouro', 'prata', 'terra'}
```

[69]:
```
tupla_de_minerios
```

[69]:
```
('ferro', 'ouro', 'cobre', 'prata', 'chumbo', 'terra')
```

[65]:
```
dicionario_de_minerios # Trabalha com o conceito de Chave e Valor
```

[65]:
```
{'minerio_1': 'ferro', 'minerio_2': 'ouro', 'lixo_1': 'terra'}
```

Ao decorrer do Notebook vai ser abordado em mais detalhes.

3.2 1.2. Operaçoes

Tinha um '+' ali no meio entre duas Strings certo, esse + serve para "concatenação" de duas Strings, em outra palavras serve para juntar essas duas palavras. Mas em matemática, serve para somar números, mesma coisa em programação

3.2.1 1.2.1. Aritméticas

[44]:
```
# A + B

1 + 1
```

[44]:
```
2
```

[46]:
```
# A - B

1 - 1
```

[46]:
```
0
```

[45]:
```
# A dividido por B

50 / 5
```

[45]:
```
10.0
```

[49]:
```
# A multiplicado por B

5 * 10
```

[49]:
```
50
```

[54]:
```
# Resultado da Divisão de A por B tem Resto?

5 % 2
```

[54]:
```
1
```

[56]:
```
5 / 2 # Realmente tem resto
```

[56]:
```
2.5
```

[48]:
```
# Resultado Interio da divisão de A por B

5 // 2
```

[48]:
```
2
```

[50]:
```
# A elevado ao expoente B

4 ** 2
```

[50]:
```
16
```

[91]:
```
# Aqui funciona as regras de sinais e a forma de esquever uam equação␣
↪matemática:

print( 'Resultado:', 3 - 5 )
print( 'Resultado:', 5 - 3 )
```

```
Resultado: -2
Resultado: 2
```

Concatenação de Strings

[57]:
```
'Olá' + 'Mundo'
```

[57]:
```
'OláMundo'
```

[60]:
```
mundo = 'Mundo'

'Ola' + ' ' + mundo
```

[60]:
```
'Ola Mundo'
```

[61]:
```
'Ola ' + 'Mundo'
```

[61]:
```
'Ola Mundo'
```

Concatenação de Listas

[71]:
```
['terra'] + ['ferro', 'chumbo', 'ouro'] + ['abacaxi']
```

[71]:
```
['terra', 'ferro', 'chumbo', 'ouro', 'abacaxi']
```

Existem outras formas de concatenar listas e trabalhar com elas, ao decorrer do Notebook vai ser abordado outras formas.

3.2.2 1.2.2. Atribuição

Lembra do = ? Ele serve para atribuir um determinado dado a alguma variavel Existem outras formas de atribuir dados no Python, formas mais 'classicas' que vem da linguagem C.

[86]:
```
a = 5

a += a

a
```

[86]: 10

[87]:
```
a = 5

a = a + 5

a
```

[87]: 10

[88]:
```
b = 5

b -= 5

b
```

[88]: 0

[89]:
```
b = 5

b = b - 5

b
```

[89]: 0

3.3 1.3. Strings na Lingaugem Python

Antes de prosseguir para o estudo das strings, recomendo analisar a documentação: https://docs.python.org/pt-br/3/library/string.html?highlight=string#module-string https://docs.python.org/pt-br/3/library/stdtypes.html#string-methods

Existem muitas formas de trabalhar com Strings, algumas dessas formas eu ja mostrei antes nesse notebook. Se você digitar help(tipo_do_dado) você vera vários métodos que podem ser aplicadas a Strings. Repare que os metodos começam depois do ___ str ___

[95]:
```
help(str)
```

```
Help on class str in module builtins:

class str(object)
 |  str(object='') -> str
 |  str(bytes_or_buffer[, encoding[, errors]]) -> str
 |
 |  Create a new string object from the given object. If encoding or
 |  errors is specified, then the object must expose a data buffer
 |  that will be decoded using the given encoding and error handler.
 |  Otherwise, returns the result of object.__str__() (if defined)
 |  or repr(object).
 |  encoding defaults to sys.getdefaultencoding().
 |  errors defaults to 'strict'.
 |
 |  Methods defined here:
 |
 |  __add__(self, value, /)
 |      Return self+value.
 |
 |  __contains__(self, key, /)
 |      Return key in self.
 |
 |  __eq__(self, value, /)
 |      Return self==value.
 |
 |  __format__(self, format_spec, /)
 |      Return a formatted version of the string as described by format_spec.
 |
 |  __ge__(self, value, /)
 |      Return self>=value.
 |
 |  __getattribute__(self, name, /)
 |      Return getattr(self, name).
 |
 |  __getitem__(self, key, /)
 |      Return self[key].
 |
 |  __getnewargs__(…)
 |
 |  __gt__(self, value, /)
 |      Return self>value.
 |
 |  __hash__(self, /)
 |      Return hash(self).
 |
 |  __iter__(self, /)
 |      Implement iter(self).
 |
```

```
 |  __le__(self, value, /)
 |      Return self<=value.
 |
 |  __len__(self, /)
 |      Return len(self).
 |
 |  __lt__(self, value, /)
 |      Return self<value.
 |
 |  __mod__(self, value, /)
 |      Return self%value.
 |
 |  __mul__(self, value, /)
 |      Return self*value.
 |
 |  __ne__(self, value, /)
 |      Return self!=value.
 |
 |  __repr__(self, /)
 |      Return repr(self).
 |
 |  __rmod__(self, value, /)
 |      Return value%self.
 |
 |  __rmul__(self, value, /)
 |      Return value*self.
 |
 |  __sizeof__(self, /)
 |      Return the size of the string in memory, in bytes.
 |
 |  __str__(self, /)
 |      Return str(self).
 |
 |  capitalize(self, /)
 |      Return a capitalized version of the string.
 |
 |      More specifically, make the first character have upper case and the rest
lower
 |      case.
 |
 |  casefold(self, /)
 |      Return a version of the string suitable for caseless comparisons.
 |
 |  center(self, width, fillchar=' ', /)
 |      Return a centered string of length width.
 |
 |      Padding is done using the specified fill character (default is a space).
 |
```

```
 |  count(…)
 |      S.count(sub[, start[, end]]) -> int
 |
 |      Return the number of non-overlapping occurrences of substring sub in
 |      string S[start:end].  Optional arguments start and end are
 |      interpreted as in slice notation.
 |
 |  encode(self, /, encoding='utf-8', errors='strict')
 |      Encode the string using the codec registered for encoding.
 |
 |      encoding
 |        The encoding in which to encode the string.
 |      errors
 |        The error handling scheme to use for encoding errors.
 |        The default is 'strict' meaning that encoding errors raise a
 |        UnicodeEncodeError.  Other possible values are 'ignore', 'replace' and
 |        'xmlcharrefreplace' as well as any other name registered with
 |        codecs.register_error that can handle UnicodeEncodeErrors.
 |
 |  endswith(…)
 |      S.endswith(suffix[, start[, end]]) -> bool
 |
 |      Return True if S ends with the specified suffix, False otherwise.
 |      With optional start, test S beginning at that position.
 |      With optional end, stop comparing S at that position.
 |      suffix can also be a tuple of strings to try.
 |
 |  expandtabs(self, /, tabsize=8)
 |      Return a copy where all tab characters are expanded using spaces.
 |
 |      If tabsize is not given, a tab size of 8 characters is assumed.
 |
 |  find(…)
 |      S.find(sub[, start[, end]]) -> int
 |
 |      Return the lowest index in S where substring sub is found,
 |      such that sub is contained within S[start:end].  Optional
 |      arguments start and end are interpreted as in slice notation.
 |
 |      Return -1 on failure.
 |
 |  format(…)
 |      S.format(*args, **kwargs) -> str
 |
 |      Return a formatted version of S, using substitutions from args and
kwargs.
 |      The substitutions are identified by braces ('{' and '}').
 |
```

```
 |  format_map(…)
 |      S.format_map(mapping) -> str
 |
 |      Return a formatted version of S, using substitutions from mapping.
 |      The substitutions are identified by braces ('{' and '}').
 |
 |  index(…)
 |      S.index(sub[, start[, end]]) -> int
 |
 |      Return the lowest index in S where substring sub is found,
 |      such that sub is contained within S[start:end].  Optional
 |      arguments start and end are interpreted as in slice notation.
 |
 |      Raises ValueError when the substring is not found.
 |
 |  isalnum(self, /)
 |      Return True if the string is an alpha-numeric string, False otherwise.
 |
 |      A string is alpha-numeric if all characters in the string are alpha-
numeric and
 |      there is at least one character in the string.
 |
 |  isalpha(self, /)
 |      Return True if the string is an alphabetic string, False otherwise.
 |
 |      A string is alphabetic if all characters in the string are alphabetic
and there
 |      is at least one character in the string.
 |
 |  isascii(self, /)
 |      Return True if all characters in the string are ASCII, False otherwise.
 |
 |      ASCII characters have code points in the range U+0000-U+007F.
 |      Empty string is ASCII too.
 |
 |  isdecimal(self, /)
 |      Return True if the string is a decimal string, False otherwise.
 |
 |      A string is a decimal string if all characters in the string are decimal
and
 |      there is at least one character in the string.
 |
 |  isdigit(self, /)
 |      Return True if the string is a digit string, False otherwise.
 |
 |      A string is a digit string if all characters in the string are digits
and there
 |      is at least one character in the string.
```

```
 |
 |  isidentifier(self, /)
 |      Return True if the string is a valid Python identifier, False otherwise.
 |
 |      Call keyword.iskeyword(s) to test whether string s is a reserved
identifier,
 |      such as "def" or "class".
 |
 |  islower(self, /)
 |      Return True if the string is a lowercase string, False otherwise.
 |
 |      A string is lowercase if all cased characters in the string are
lowercase and
 |      there is at least one cased character in the string.
 |
 |  isnumeric(self, /)
 |      Return True if the string is a numeric string, False otherwise.
 |
 |      A string is numeric if all characters in the string are numeric and
there is at
 |      least one character in the string.
 |
 |  isprintable(self, /)
 |      Return True if the string is printable, False otherwise.
 |
 |      A string is printable if all of its characters are considered printable
in
 |      repr() or if it is empty.
 |
 |  isspace(self, /)
 |      Return True if the string is a whitespace string, False otherwise.
 |
 |      A string is whitespace if all characters in the string are whitespace
and there
 |      is at least one character in the string.
 |
 |  istitle(self, /)
 |      Return True if the string is a title-cased string, False otherwise.
 |
 |      In a title-cased string, upper- and title-case characters may only
 |      follow uncased characters and lowercase characters only cased ones.
 |
 |  isupper(self, /)
 |      Return True if the string is an uppercase string, False otherwise.
 |
 |      A string is uppercase if all cased characters in the string are
uppercase and
 |      there is at least one cased character in the string.
```

```
 |
 |  join(self, iterable, /)
 |      Concatenate any number of strings.
 |
 |      The string whose method is called is inserted in between each given
string.
 |      The result is returned as a new string.
 |
 |      Example: '.'.join(['ab', 'pq', 'rs']) -> 'ab.pq.rs'
 |
 |  ljust(self, width, fillchar=' ', /)
 |      Return a left-justified string of length width.
 |
 |      Padding is done using the specified fill character (default is a space).
 |
 |  lower(self, /)
 |      Return a copy of the string converted to lowercase.
 |
 |  lstrip(self, chars=None, /)
 |      Return a copy of the string with leading whitespace removed.
 |
 |      If chars is given and not None, remove characters in chars instead.
 |
 |  partition(self, sep, /)
 |      Partition the string into three parts using the given separator.
 |
 |      This will search for the separator in the string.  If the separator is
found,
 |      returns a 3-tuple containing the part before the separator, the
separator
 |      itself, and the part after it.
 |
 |      If the separator is not found, returns a 3-tuple containing the original
string
 |      and two empty strings.
 |
 |  removeprefix(self, prefix, /)
 |      Return a str with the given prefix string removed if present.
 |
 |      If the string starts with the prefix string, return
string[len(prefix):].
 |      Otherwise, return a copy of the original string.
 |
 |  removesuffix(self, suffix, /)
 |      Return a str with the given suffix string removed if present.
 |
 |      If the string ends with the suffix string and that suffix is not empty,
 |      return string[:-len(suffix)]. Otherwise, return a copy of the original
```

```
 |      string.
 |
 |  replace(self, old, new, count=-1, /)
 |      Return a copy with all occurrences of substring old replaced by new.
 |
 |        count
 |          Maximum number of occurrences to replace.
 |          -1 (the default value) means replace all occurrences.
 |
 |      If the optional argument count is given, only the first count
occurrences are
 |      replaced.
 |
 |  rfind(…)
 |      S.rfind(sub[, start[, end]]) -> int
 |
 |      Return the highest index in S where substring sub is found,
 |      such that sub is contained within S[start:end].  Optional
 |      arguments start and end are interpreted as in slice notation.
 |
 |      Return -1 on failure.
 |
 |  rindex(…)
 |      S.rindex(sub[, start[, end]]) -> int
 |
 |      Return the highest index in S where substring sub is found,
 |      such that sub is contained within S[start:end].  Optional
 |      arguments start and end are interpreted as in slice notation.
 |
 |      Raises ValueError when the substring is not found.
 |
 |  rjust(self, width, fillchar=' ', /)
 |      Return a right-justified string of length width.
 |
 |      Padding is done using the specified fill character (default is a space).
 |
 |  rpartition(self, sep, /)
 |      Partition the string into three parts using the given separator.
 |
 |      This will search for the separator in the string, starting at the end.
If
 |      the separator is found, returns a 3-tuple containing the part before the
 |      separator, the separator itself, and the part after it.
 |
 |      If the separator is not found, returns a 3-tuple containing two empty
strings
 |      and the original string.
 |
```

```
 |  rsplit(self, /, sep=None, maxsplit=-1)
 |      Return a list of the words in the string, using sep as the delimiter
string.
 |
 |        sep
 |          The delimiter according which to split the string.
 |          None (the default value) means split according to any whitespace,
 |          and discard empty strings from the result.
 |        maxsplit
 |          Maximum number of splits to do.
 |          -1 (the default value) means no limit.
 |
 |      Splits are done starting at the end of the string and working to the
front.
 |
 |  rstrip(self, chars=None, /)
 |      Return a copy of the string with trailing whitespace removed.
 |
 |      If chars is given and not None, remove characters in chars instead.
 |
 |  split(self, /, sep=None, maxsplit=-1)
 |      Return a list of the words in the string, using sep as the delimiter
string.
 |
 |      sep
 |        The delimiter according which to split the string.
 |        None (the default value) means split according to any whitespace,
 |        and discard empty strings from the result.
 |      maxsplit
 |        Maximum number of splits to do.
 |        -1 (the default value) means no limit.
 |
 |  splitlines(self, /, keepends=False)
 |      Return a list of the lines in the string, breaking at line boundaries.
 |
 |      Line breaks are not included in the resulting list unless keepends is
given and
 |      true.
 |
 |  startswith(…)
 |      S.startswith(prefix[, start[, end]]) -> bool
 |
 |      Return True if S starts with the specified prefix, False otherwise.
 |      With optional start, test S beginning at that position.
 |      With optional end, stop comparing S at that position.
 |      prefix can also be a tuple of strings to try.
 |
 |  strip(self, chars=None, /)
```

```
 |      Return a copy of the string with leading and trailing whitespace
removed.
 |
 |      If chars is given and not None, remove characters in chars instead.
 |
 |  swapcase(self, /)
 |      Convert uppercase characters to lowercase and lowercase characters to
uppercase.
 |
 |  title(self, /)
 |      Return a version of the string where each word is titlecased.
 |
 |      More specifically, words start with uppercased characters and all
remaining
 |      cased characters have lower case.
 |
 |  translate(self, table, /)
 |      Replace each character in the string using the given translation table.
 |
 |        table
 |          Translation table, which must be a mapping of Unicode ordinals to
 |          Unicode ordinals, strings, or None.
 |
 |      The table must implement lookup/indexing via __getitem__, for instance a
 |      dictionary or list.  If this operation raises LookupError, the character
is
 |      left untouched.  Characters mapped to None are deleted.
 |
 |  upper(self, /)
 |      Return a copy of the string converted to uppercase.
 |
 |  zfill(self, width, /)
 |      Pad a numeric string with zeros on the left, to fill a field of the
given width.
 |
 |      The string is never truncated.
 |
 |  ----------------------------------------------------------------------
 |  Static methods defined here:
 |
 |  __new__(*args, **kwargs) from builtins.type
 |      Create and return a new object.  See help(type) for accurate signature.
 |
 |  maketrans(…)
 |      Return a translation table usable for str.translate().
 |
 |      If there is only one argument, it must be a dictionary mapping Unicode
 |      ordinals (integers) or characters to Unicode ordinals, strings or None.
```

```
|      Character keys will be then converted to ordinals.
|      If there are two arguments, they must be strings of equal length, and
|      in the resulting dictionary, each character in x will be mapped to the
|      character at the same position in y. If there is a third argument, it
|      must be a string, whose characters will be mapped to None in the result.
```

Vamos Testar Alguns.

Resumidamente a Primeira letra é maiuscula e as demais minusculas.

[96]:
```
string = 'ola mundo'
```

[97]:
```
string.capitalize()
```

[97]:
```
'Ola mundo'
```

[3]:
```
# Observação: Caso você não coloque o ''()'' no final.
# Você vai retornar a função, não o resultado da função

'ola mundo'.capitalize
```

[3]:
```
<function str.capitalize()>
```

[99]:
```
'oLA MUNDO'.capitalize()
```

[99]:
```
'Ola mundo'
```

Vamos testar alguns a mais

[101]:
```
print( string.upper() ) # Maiuscula
print( string.lower() ) # Minuscdula
print( string.title() ) # Título Maiusculo
```

```
OLA MUNDO
ola mundo
Ola Mundo
```

Existem até funções que retornam dados boleanos para futuras checagens como o 'endswith()'

[102]:
```
'Ola Mundo'.endswith('o')
```

[102]:
```
True
```

[103]:
```
'Ola Mundo'.startswith('o') # Lembra que Maiusculo e Minusculo faz diferença
```

[103]:
```
False
```

Também é possível modificar essas strings e cortar pedaços dela, por exemplo

[104]:
```
# Trocar espaços por Underscore

'Ola Mundo Doido'.replace(' ', '_')
```

[104]:
```
'Ola_Mundo_Doido'
```

[109]:
```
# Repare que o tipo de dado é STRING, logo eu ainda posso aplciar mais métodos␣
 ↪nessa string.
# A sequencia começa da esquerda para a direita

'Ola Mundo Doido'.replace(' ', '_').lower()
```

[109]:
```
'ola_mundo_doido'
```

[111]:
```
'Ola Mundo Doido'.replace(' ', '_').lower().replace('_', '-').title()
```

[111]:
```
'Ola-Mundo-Doido'
```

[114]:
```
# Nesse caso já foi alterado o tipo de dado, agora não é mais STRING, mais sim␣
 ↪uma Lista

lista_doida = 'Ola Mundo Doido'.replace(' ', '_').lower().replace('_', '-').
 ↪title().split('-')
lista_doida
```

[114]:
```
['Ola', 'Mundo', 'Doido']
```

[115]:
```
type(lista_doida)
```

[115]:
```
list
```

É possivel utilizar o conceito de Slicing e pegar pedaços da String, por exemplo

[132]:
```
string = 'Hoje eu vou dormir mais cedo'

print( 'Primeira Letra: ',    string[0] )
print( 'Ultima Letra: ',      string[-1] )
print( 'Algumas Letras: ',    string[0:4] )
print( 'Penultima Palavra: ', string[-9:-5] )

'''
Reparem que a seleção é feita pelos índices das Letras

String:    A Casa
índice:    012345

'''
```

```
Primeira Letra:  H
Ultima Letra:  o
Algumas Letras:  Hoje
Penultima Palavra:  mais
```

[134]:
```
print( 'A Casa'[1] ) # é o Espaço
print( 'A Casa'[0] ) # é o A
```

```
A
```

[141]:
```
# Você tambem pode selecionar uma Sequencia de Onde quer cortar

print( 'A Casa'[0:] )
print( 'A Casa'[2:] )
print( 'A Casa'[:-1] )
```

```
A Casa
Casa
A Cas
```

3.4 1.4. Convertendo dados com Python

Para converter dados em python é bem simples, existem funções nativas que fazer esse trabalho. int() Converte para Inteiro float() converte para Float str() converte para String...

[9]:
```
dez = '10'

dez_int = int( dez ) # Converter o '10' em 10

print(dez)
print(dez_int)
```

```
10
10
```

[11]:
```
print(type(dez))
print(type(dez_int))
```

```
<class 'str'>
<class 'int'>
```

[13]:
```
int('dez') # Python não sabe Português
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Input In [13], in <cell line: 1>()
----> 1 int('dez')
```

```
ValueError: invalid literal for int() with base 10: 'dez'
```

4 2.0. Estruturas Condicionais

As Estruturas condicionais servem basicamente para aplicar condições no seu código, para ele não ser linear.

Existe alguns Operadores também para fazer essa checagem:

(A == B) Verifica se A é igual a B (A >= B) Verifica se A é maior e igual que B (A <= B) Verifica se A é menor ou igual a B (A != B) Verifica se A é diferente de B

[6]:
```
a = 5
b = 10

#SE A é Igual a 5, então:
    # ...

if a == 5:
    print('A é Igual a 5')

if a != b:
    print('A é diferente de B')
```

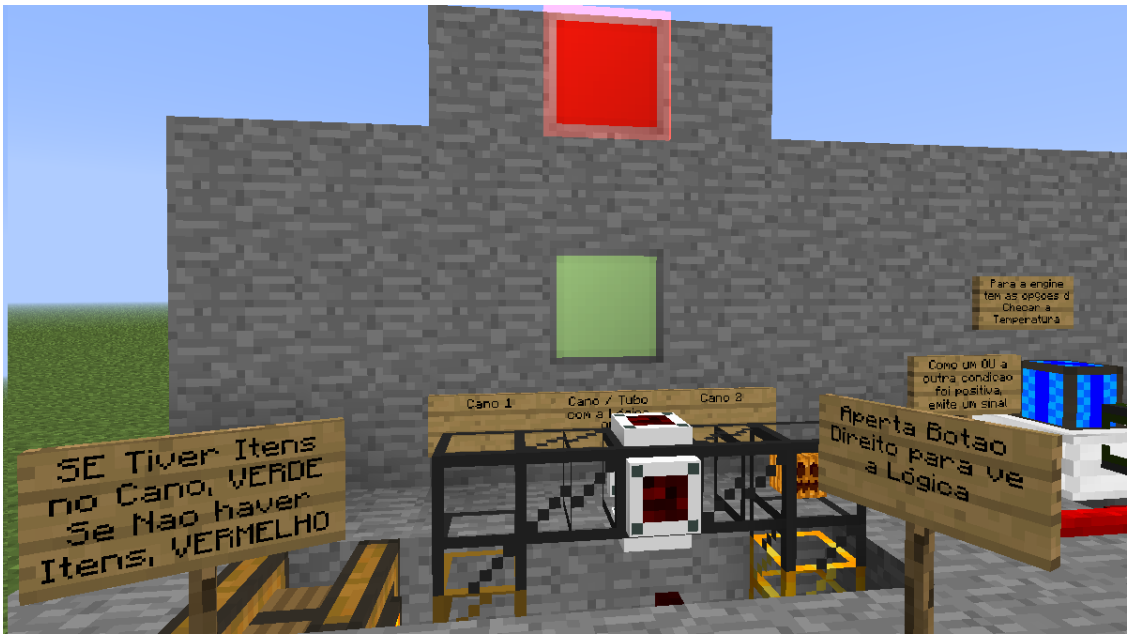
```
A é Igual a 5
A é diferente de B
```

Repare que é necessário dois pontos no final para indicar o **IF** Também o Python automaticamente prepara a indentação ou seja, os 4 espaços, indicando que ali realmente vai ser a execução do IF.

Pode também ser checado mais que uma condição com os operadores E | Ou.

[5]:
```
Image('../n_img/mine0.png')
```

[5]:



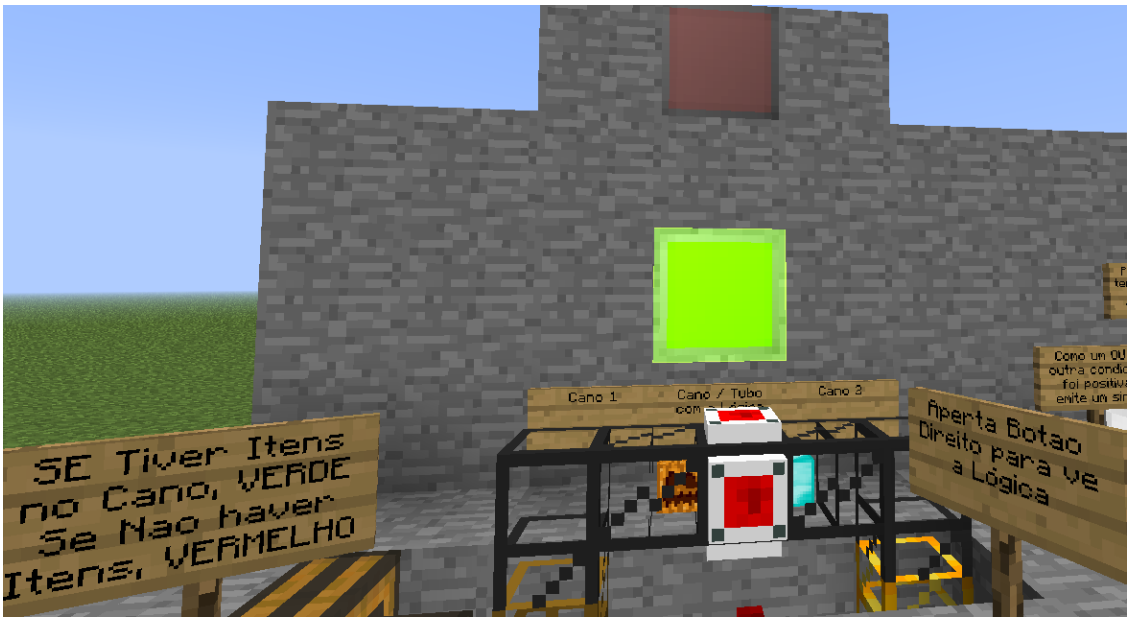
Se tiver itens no Cano, então Verde, se não haver itens, Vermelho

[7]:
```
itens_no_cano = False

if itens_no_cano:
    print('VERDE')

else:
    print('VERMELHO')
```

```
VERMELHO
```

[8]:
```
Image('../n_img/mine1.png')
```

[8]:



[10]:
```
itens_no_cano = True

if itens_no_cano:
    print('VERDE')

else:
    print('VERMELHO')
```

```
VERDE
```

Podemos fazer de outras formas a mesma checagem

[20]:
```
# Se existir Itens e Existir itens no Cano, então Verde
# Senão, vermelho

itens = True
cano_com_itens = False

if itens:
    if cano_com_itens:
        print('VERDE')
    else:
        print('VERMELHO')
```

```
VERMELHO
```

[23]:
```
# Se existir Itens e Existir itens no Cano, então Verde
# Senão, vermelho
```

```
itens = True
cano_com_itens = True

if itens == True:
    if cano_com_itens:
        print('VERDE')
    else:
        print('VERMELHO')
```

```
VERDE
```

O operador AND checa se duas coisas são verdadeiras, olhe o exemplo abaixo:

[22]:
```
Image('../n_img/mine3.png')
```

[22]:

[24]:
```
# No exemplo Anterior, ficarria assim:
# Existem Itens & Os itens estão no Cano

itens = True
cano_com_itens = True

if (itens) & (cano_com_itens):
    print('VERDE')
else:
    print('VERMELHO')
```

```
VERDE
```

[29]:
```
# Se a quantidade de carvao for maior que 2 e o Motor for azul, então executa o␣
↪motor
```

```
carvao = 3
motor = 'azul'

if carvao >= 2:
    if motor != 'azul':
        print('Para o Motor')
    else:
        print('Executa o Motor')
```

```
Executa o Motor
```

Porem, é possível melhorar isso, o carvao esta realmente dentro do motor ou o motor esta ligado pelo menos?

[33]:
```
alavanca = True
qntd_carvao = 5
combustivel = 'carvao_coque'
aquecimento_motor = 'azul'
combustivel_motor = {'carvao', 'madeira', 'carvao_coque', 'carvao_vegetal',␣
 ↪'carvao_mineral'}

if combustivel in combustivel_motor: # Se o combustível esta no motor
    print('[OK] Combustível Valido para o Motor')

    if qntd_carvao >= 2:
        print(f'[OK] Quantidade Ideal de {combustivel}')

        if aquecimento_motor == 'azul':
            print('[OK] A coloração do motor é Ideal')

            if alavanca:
                print('-> Alavanca Acionada, motor em Funcionamento')

            else:
                print('-> Tudo Pronto para o Funcionamento do Motor, basta␣
 ↪Ligar')

        else:
            print(f'[ERRO] O aquecimento é da cor: {aquecimento_motor}')

    else:
        print('[ERRO] Não há uma quantidade Ideal para o funcionamento do␣
 ↪Motor')

else:
    print('[ERRO] Combustível Invalido para o Motor')
```

```
[OK] Combustível Valido para o Motor
```

```
[OK] Quantidade Ideal de carvao_coque
[OK] A coloração do motor é Ideal
-> Alavanca Acionada, motor em Funcionamento
```

É possível Simplificar esse código.

[54]:
```
alavanca = True
qntd_carvao = 1
combustivel = 'carvao_coque'
aquecimento_motor = 'azul'
combustivel_motor = {'carvao', 'madeira', 'carvao_coque', 'carvao_vegetal',␣
 ↪'carvao_mineral'}

if combustivel in combustivel_motor:
    print('[OK] Combustível Valido para o Motor')

    if (qntd_carvao >= 2) & (aquecimento_motor == 'azul') & (alavanca):
        print(f'[OK] Motor Pronto para Funcionar')

    else:
        if alavanca:
            alavanca = 'Ligada'
        else:
            alavanca = 'Desligada'

        print(f'[ERRO] Motor não está devidamente preparado, Valide: \n\
       Aquecimento: | {aquecimento_motor} | Quantidade de Carvão: |␣
 ↪{qntd_carvao} | Alavanca: {alavanca}')

else:
    print('[ERRO] Combustível Invalido para o Motor')
```

```
[OK] Combustível Valido para o Motor
[ERRO] Motor não está devidamente preparado, Valide:
       Aquecimento: | azul | Quantidade de Carvão: | 1 | Alavanca: Ligada
```

4.1 2.2. Exemplo da Quary

Lembra do Primeiro Exemplo de introdução, então vamos fazer um programinha para ele: “Você foi desafiado a construir um esquema de Mineradora automatica no Minecraft, precisa definir todo o esquema de tempo de entrega de itens para minimizar o consumo de energia das máquinas.”

Objetivos:

1. Definir a Mineradora. - Existe A Mineradora? - Combustível Ideal é 10.0 L, sendo Petróleo ou Gasolina. - Aquecimento da Quary tem que ser igual a azul, verde ou amarelo.
2. Definir o Armazenamento dos Itens. - Canos especificos para o Transporte, Minimo 5. - Canos de Ouro para acelerar o Transporte, Redução do Tempo de 15 segundos - Capacidade de Armazenamento é igual a 50

3. Definir o Formato de Transporte de Itens. - Normal = 10, Ouro = 2
4. Definir a Entrega dos Pós de Minerios. - Onde vou entregar os Minerios processados?

[91]:
```
# Definição da Mineradora

quary = True
combustivel_quary  = '15.6, Gasolina'
aquecimento_query  = ['azul', 'verde', 'amarelo']
litros_minimos     = 10
aquecimento_minimo = 'azul'
minera_por_segundo = 1

# Definição Armazenamento

capacidade  = 50
armazenados = 32
tempo_para_armazenar = '33s'

# Definição dos Tubos

normal_tempo = 0
ouro_tempo   = 15
```

[92]:
```
import time
combustivel_quary_litros = float(combustivel_quary.split(',')[0])
combustivel_quary_tipo   = combustivel_quary.split(',')[1].strip()
tempo_para_armazenar     = int(tempo_para_armazenar[:-1])

'''
Antes de Ligar a Quary, Verificar:
 - Se Tem espaço para Armazenar
 - Se o tempo de Armazenar é Menor que o Tempo da Mineração

'''

if (armazenados <= capacidade) & (minera_por_segundo >= tempo_para_armazenar):
    print('[ERRO] Verifique a Capacidade de Armazenamento ou Tempo para␣
 ↪Armazenar')
else:
    # Com três canos de ouro, o tempo de armazenamento é menor que a coleta
    qntd_canos_ouro = tempo_para_armazenar - ( ouro_tempo * 3 )

    if minera_por_segundo <= tempo_para_armazenar:
        print('[OK] É possível Armazenar Itens mais rapido que a Coleta')

        combustiveis_aceitos = ['Gasolina', 'Petroleo']
```

```
        if (aquecimento_minimo in aquecimento_query) & \
           (litros_minimos <= combustivel_quary_litros) & \
           (combustivel_quary_tipo in combustiveis_aceitos):

            print('[OK] Quary Em Funcionamento...')

            time.sleep(5)

            print('[OK] Quary Acabou de Coletar os Minerios')

            itens_descatar  = ['Terra', 'Areia', 'Cascalho']
            itens_coletados = ['Ferro', 'Ouro', 'Terra', 'Cascalho']

            for itens in itens_coletados:
                if not itens in itens_descatar:
                    print('-> Item Prosseguiu ao Processamento')
        else:
            print('[ERROR] Verifique a Quary e Tente Novamente')

    else:
        print('[ERRO] É Necessário mais tempo para Armazenar')
```

```
[OK] É possível Armazenar Itens mais rapido que a Coleta
[OK] Quary Em Funcionamento…
[OK] Quary Acabou de Coletar os Minerios
-> Item Prosseguiu ao Processamento
-> Item Prosseguiu ao Processamento
```

No futuro esse exemplo vai ser melhorado, por enquanto, tem algumas coisas ali que vamos ver ao decorrer do Notebook, como o For, mas antes, vamos verificar como as listas funcionam.

5 3.0. Laços de Repetição

Os laços de repetição nada mais são de uma forma de rodar várias vezes a mesma iteração de um determinado passo. No exemplo da Quary, foi utilizado para obter os itens coletados pela quary.

O FOR pega cada elemento de uma lista de tamanho n e executa n vezes uma determinada função, pode ser utilizada cada elemento da lista para fazer alguma comparação ou simplesmente rodar n vezes o comando abaixo do for.

[1]:
```
# O range é uma função que cria uma lista de Números de um N° Inicial ate um␣
↪Certo Número começando em 0.
list(range(0, 4))
```

[1]:
```
[0, 1, 2, 3]
```

[2]:
```
# Para cada i em uma lista de 0 a 3, executa:

for i in range( 0, 4 ):
    print(i)
```

```
0
1
2
3
```

[3]:
```
for i in range( 0, 4 ):
    print(f'Executar o Print pela {i+1}° Vez')
```

```
Executar o Print pela 1° Vez
Executar o Print pela 2° Vez
Executar o Print pela 3° Vez
Executar o Print pela 4° Vez
```

[4]:
```
# Enquanto o i for menor ou igual a 4, executa o Print

i = 0
while i <= 4:
    print(f'Executar o Print pela {i+1}° Vez')
    i += 1
```

```
Executar o Print pela 1° Vez
Executar o Print pela 2° Vez
Executar o Print pela 3° Vez
Executar o Print pela 4° Vez
Executar o Print pela 5° Vez
```

[5]:
```
minerios = ['Ouro', 'Ferro', 'Terra', 'Cascalho']

# Para cada Minerio da Lista de Minerios, printa eles
for minerio in minerios:
    print(minerio)
```

```
Ouro
Ferro
Terra
Cascalho
```

[6]:
```
minerios = ['Ouro', 'Ferro', 'Terra', 'Cascalho']

itens_para_descartar = ['Cascalho', 'Terra']

# Para Cada Minério na lista de Minérios
    # Verifica se o Minerio vai ser Descartado ou Não
```

```
for minerio in minerios:
    if minerio in itens_para_descartar:
        print(f'Minerio Descartado: {minerio}')

    else:
        print(f'Minério Enviado: {minerio}')
```

```
Minério Enviado: Ouro
Minério Enviado: Ferro
Minerio Descartado: Terra
Minerio Descartado: Cascalho
```

5.1 3.1. Listas em Python

Uma lista ou Vetor nada mais é que uma variável com Vários dados :D

Você pode aplicar diversas transformações como 'Fatiar' ou cortar um pedaço da lista como foi visualizado nas Strings, pode também aplicar outros n funções em uma lista atravez do operador **.**

[7]:
```
help(list)
```

```
Help on class list in module builtins:

class list(object)
 |  list(iterable=(), /)
 |
 |  Built-in mutable sequence.
 |
 |  If no argument is given, the constructor creates a new empty list.
 |  The argument must be an iterable if specified.
 |
 |  Methods defined here:
 |
 |  __add__(self, value, /)
 |      Return self+value.
 |
 |  __contains__(self, key, /)
 |      Return key in self.
 |
 |  __delitem__(self, key, /)
 |      Delete self[key].
 |
 |  __eq__(self, value, /)
 |      Return self==value.
 |
 |  __ge__(self, value, /)
 |      Return self>=value.
 |
```

```
|  __getattribute__(self, name, /)
|      Return getattr(self, name).
|
|  __getitem__(…)
|      x.__getitem__(y) <==> x[y]
|
|  __gt__(self, value, /)
|      Return self>value.
|
|  __iadd__(self, value, /)
|      Implement self+=value.
|
|  __imul__(self, value, /)
|      Implement self*=value.
|
|  __init__(self, /, *args, **kwargs)
|      Initialize self.  See help(type(self)) for accurate signature.
|
|  __iter__(self, /)
|      Implement iter(self).
|
|  __le__(self, value, /)
|      Return self<=value.
|
|  __len__(self, /)
|      Return len(self).
|
|  __lt__(self, value, /)
|      Return self<value.
|
|  __mul__(self, value, /)
|      Return self*value.
|
|  __ne__(self, value, /)
|      Return self!=value.
|
|  __repr__(self, /)
|      Return repr(self).
|
|  __reversed__(self, /)
|      Return a reverse iterator over the list.
|
|  __rmul__(self, value, /)
|      Return value*self.
|
|  __setitem__(self, key, value, /)
|      Set self[key] to value.
|
```

```
 |  __sizeof__(self, /)
 |      Return the size of the list in memory, in bytes.
 |
 |  append(self, object, /)
 |      Append object to the end of the list.
 |
 |  clear(self, /)
 |      Remove all items from list.
 |
 |  copy(self, /)
 |      Return a shallow copy of the list.
 |
 |  count(self, value, /)
 |      Return number of occurrences of value.
 |
 |  extend(self, iterable, /)
 |      Extend list by appending elements from the iterable.
 |
 |  index(self, value, start=0, stop=9223372036854775807, /)
 |      Return first index of value.
 |
 |      Raises ValueError if the value is not present.
 |
 |  insert(self, index, object, /)
 |      Insert object before index.
 |
 |  pop(self, index=-1, /)
 |      Remove and return item at index (default last).
 |
 |      Raises IndexError if list is empty or index is out of range.
 |
 |  remove(self, value, /)
 |      Remove first occurrence of value.
 |
 |      Raises ValueError if the value is not present.
 |
 |  reverse(self, /)
 |      Reverse *IN PLACE*.
 |
 |  sort(self, /, *, key=None, reverse=False)
 |      Sort the list in ascending order and return None.
 |
 |      The sort is in-place (i.e. the list itself is modified) and stable (i.e.
the
 |      order of two equal elements is maintained).
 |
 |      If a key function is given, apply it once to each list item and sort
them,
```

```
|      ascending or descending, according to their function values.
|
|      The reverse flag can be set to sort in descending order.
|
|  ----------------------------------------------------------------------
|  Class methods defined here:
|
|  __class_getitem__(…) from builtins.type
|      See PEP 585
|
|  ----------------------------------------------------------------------
|  Static methods defined here:
|
|  __new__(*args, **kwargs) from builtins.type
|      Create and return a new object.  See help(type) for accurate signature.
|
|  ----------------------------------------------------------------------
|  Data and other attributes defined here:
|
|  __hash__ = None
```

[8]:
```
lista = [1, 2, 3, 'pão_de_cima', 'queijo', 'presunto', 'pão_de_baixo']

lista
```

[8]:
```
[1, 2, 3, 'pão_de_cima', 'queijo', 'presunto', 'pão_de_baixo']
```

[9]:
```
# Tutorial como Fazer um Sanduiche em Python:

ingredientes = lista[3:]
ingredientes
```

[9]:
```
['pão_de_cima', 'queijo', 'presunto', 'pão_de_baixo']
```

[10]:
```
# O reversed retorna um Iterador, é necessário transformar em lista novamente.
# Ao decorrer do notebook eu vou explicar direitinho oque ele significa, mas␣
 ↪resumidamente ele inverte a lista.

ingredientes = list( reversed( ingredientes ) )
ingredientes
```

[10]:
```
['pão_de_baixo', 'presunto', 'queijo', 'pão_de_cima']
```

[11]:
```
print(f'Primeiro corta o pão em duas PARTES: {ingredientes[0]} e o␣
 ↪{ingredientes[-1]} do Sanduiche')
print('Depois monta nessa sequencia: \n')
```

```
Primeiro corta o pão em duas PARTES: pão_de_baixo e o pão_de_cima do Sanduiche
Depois monta nessa sequencia:
```

[12]:
```python
if ingredientes[0] == 'pão_de_baixo':
    # Python não aceita duas aspas, utilize aspas duplas e simples
    print(f"Começa com o | {ingredientes[0].replace('_', ' ').title()} |")

    print('\n -> Seguido dos demais ingredintes: \n')

    i = 1
    for ingred in ingredientes[1:]:
        print(f"{i}° Ingredinte é {ingred.replace('_', ' ').title()}")
        i += 1
```

```
Começa com o | Pão De Baixo |

 -> Seguido dos demais ingredintes:

1° Ingredinte é Presunto
2° Ingredinte é Queijo
3° Ingredinte é Pão De Cima
```

Existem várias formas mais elaboradas de fazer esse sanduiche com python, mas por enquanto vamos deixar dessa forma Ok? Mas agora você ja sabe como fazer um Sanduiche :D

Uma função bem legal é o append(), ela adiciona mais dados em uma lista.

Mas antes, olhe o exemplo, estou importando uma **biblioteca** chamada random, que nela existem vários **métodos**, esses métodos nada mais são que funçoes que eu posso utilizar no meu código para agilizar um processo, nesse exemplo da **biblioteca** random, estou chamando o **método** randint que gera um numero aleatorio, e nesse randint, estou passando de **argumento** dois parâmetros, o 0 e o 3, que nada mais é que a dimensão do número aleatório, só pode ser de 0 a 4.

[13]:
```python
import random

# Para cada NADA em 5 vezes KKK
# Ou em outras palavras, roda 5x o comando abaixo

for _ in range( 0, 5 ):

    print( random.randint(0, 4) )

    # Gera um resultado aleatório entre 0 a 4
```

```
3
4
0
2
4
```

Vamos ligar a Mineradora em uma região que somente tem Ouro, Ferro, Chumbo e Cobre, e deixar ela trabalhar por 50 vezes. E registrar quais Minérios ela Coletou.

[14]:
```
minerios_da_regiao = ['Ouro', 'Ferro', 'Chumbo', 'Cobre']
            # Index     0        1        2         3

# Uma lista vazia vai ser a quantidade de Minerios que a Mineradora coletou
minerios_coletados = []

for _ in range( 50 ):

    # Adiciona na lista Vazia, os Minerios Aleatoriamente pelo índice de ( 0 a␣
 ↪3 )

    minerios_coletados.append( minerios_da_regiao[random.randint(0, 3)] )
```

[15]:
```
# A função LEN imprime o comprimento de um Vetor
print( f'Tamanho do Armazem de Minérios Atualmente: {len(minerios_coletados)} ')
```

```
Tamanho do Armazem de Minérios Atualmente: 50
```

[16]:
```
minerios_coletados
```

[16]:
```
['Ouro',
 'Cobre',
 'Ouro',
 'Ouro',
 'Cobre',
 'Cobre',
 'Ferro',
 'Chumbo',
 'Cobre',
 'Cobre',
 'Cobre',
 'Ferro',
 'Ouro',
 'Chumbo',
 'Chumbo',
 'Cobre',
 'Ouro',
 'Ouro',
 'Cobre',
 'Ferro',
 'Chumbo',
 'Ferro',
 'Cobre',
 'Chumbo',
 'Chumbo',
```

```
 'Ouro',
 'Chumbo',
 'Ouro',
 'Ouro',
 'Ferro',
 'Cobre',
 'Ferro',
 'Cobre',
 'Ferro',
 'Ferro',
 'Ferro',
 'Ouro',
 'Ferro',
 'Cobre',
 'Ferro',
 'Cobre',
 'Ouro',
 'Ouro',
 'Chumbo',
 'Cobre',
 'Ouro',
 'Cobre',
 'Ouro',
 'Ouro',
 'Ferro']
```

Vamos Limpar agora caso vier Terra, Cascalho ou Areia

[17]:
```
minerios_da_regiao = ['Ouro', 'Ferro', 'Chumbo', 'Cobre', 'Terra', 'Cascalho',␣
↪'Areia']
           # Index     0        1        2        3        4          5       ␣
↪  6

# Uma lista vazia vai ser a quantidade de Minerios que a Mineradora coletou
minerios_coletados = []

for _ in range( 50 ):

    # é necessário Fixar o índice
    indice_aleatorio = random.randint(0, 6)

    # Se o Minerio Coletado não estiver na lista de minerios, então é lixo␣
↪coletado
    if minerios_da_regiao[ indice_aleatorio ] not in minerios_da_regiao[:-3]:

        # Se for lixo coletado, então não faz nada, passa para a proxima linha
        pass
```

```
    # Se não for lixo, então armazena no Armazem de Itens
    else:
        minerios_coletados.append( minerios_da_regiao[ indice_aleatorio ]  )
```

[18]:
```
# Minerios Apenas.
minerios_da_regiao[:-3]
```

[18]:
```
['Ouro', 'Ferro', 'Chumbo', 'Cobre']
```

[19]:
```
print(f'Quantidade de Minérios Coletados: {len(minerios_coletados)} | Vezes que␣
 ↪Coletou Alguma Coisa: {len(range( 50 ))}')
```

```
Quantidade de Minérios Coletados: 30 | Vezes que Coletou Alguma Coisa: 50
```

5.2 3.2. Dicionarios em Python

Dicionario é outra estrutura bem diferente, mas que armazena vários dados também.

[20]:
```
help(dict)
```

```
Help on class dict in module builtins:

class dict(object)
 |  dict() -> new empty dictionary
 |  dict(mapping) -> new dictionary initialized from a mapping object's
 |      (key, value) pairs
 |  dict(iterable) -> new dictionary initialized as if via:
 |      d = {}
 |      for k, v in iterable:
 |          d[k] = v
 |  dict(**kwargs) -> new dictionary initialized with the name=value pairs
 |      in the keyword argument list.  For example:  dict(one=1, two=2)
 |
 |  Built-in subclasses:
 |      StgDict
 |
 |  Methods defined here:
 |
 |  __contains__(self, key, /)
 |      True if the dictionary has the specified key, else False.
 |
 |  __delitem__(self, key, /)
 |      Delete self[key].
 |
 |  __eq__(self, value, /)
 |      Return self==value.
 |
 |  __ge__(self, value, /)
```

```
|      Return self>=value.
|
|  __getattribute__(self, name, /)
|      Return getattr(self, name).
|
|  __getitem__(…)
|      x.__getitem__(y) <==> x[y]
|
|  __gt__(self, value, /)
|      Return self>value.
|
|  __init__(self, /, *args, **kwargs)
|      Initialize self.  See help(type(self)) for accurate signature.
|
|  __ior__(self, value, /)
|      Return self|=value.
|
|  __iter__(self, /)
|      Implement iter(self).
|
|  __le__(self, value, /)
|      Return self<=value.
|
|  __len__(self, /)
|      Return len(self).
|
|  __lt__(self, value, /)
|      Return self<value.
|
|  __ne__(self, value, /)
|      Return self!=value.
|
|  __or__(self, value, /)
|      Return self|value.
|
|  __repr__(self, /)
|      Return repr(self).
|
|  __reversed__(self, /)
|      Return a reverse iterator over the dict keys.
|
|  __ror__(self, value, /)
|      Return value|self.
|
|  __setitem__(self, key, value, /)
|      Set self[key] to value.
|
|  __sizeof__(…)
```

```
 |      D.__sizeof__() -> size of D in memory, in bytes
 |
 |  clear(…)
 |      D.clear() -> None.  Remove all items from D.
 |
 |  copy(…)
 |      D.copy() -> a shallow copy of D
 |
 |  get(self, key, default=None, /)
 |      Return the value for key if key is in the dictionary, else default.
 |
 |  items(…)
 |      D.items() -> a set-like object providing a view on D's items
 |
 |  keys(…)
 |      D.keys() -> a set-like object providing a view on D's keys
 |
 |  pop(…)
 |      D.pop(k[,d]) -> v, remove specified key and return the corresponding
value.
 |
 |      If key is not found, default is returned if given, otherwise KeyError is
raised
 |
 |  popitem(self, /)
 |      Remove and return a (key, value) pair as a 2-tuple.
 |
 |      Pairs are returned in LIFO (last-in, first-out) order.
 |      Raises KeyError if the dict is empty.
 |
 |  setdefault(self, key, default=None, /)
 |      Insert key with a value of default if key is not in the dictionary.
 |
 |      Return the value for key if key is in the dictionary, else default.
 |
 |  update(…)
 |      D.update([E, ]**F) -> None.  Update D from dict/iterable E and F.
 |      If E is present and has a .keys() method, then does:  for k in E: D[k] =
E[k]
 |      If E is present and lacks a .keys() method, then does:  for k, v in E:
D[k] = v
 |      In either case, this is followed by: for k in F:  D[k] = F[k]
 |
 |  values(…)
 |      D.values() -> an object providing a view on D's values
 |
 |  ----------------------------------------------------------------------
 |  Class methods defined here:
```

```
|
|  __class_getitem__(…) from builtins.type
|      See PEP 585
|
|  fromkeys(iterable, value=None, /) from builtins.type
|      Create a new dictionary with keys from iterable and values set to value.
|
|  ----------------------------------------------------------------------
|  Static methods defined here:
|
|  __new__(*args, **kwargs) from builtins.type
|      Create and return a new object.  See help(type) for accurate signature.
|
|  ----------------------------------------------------------------------
|  Data and other attributes defined here:
|
|  __hash__ = None
```

[21]:
```
dicionario = {'Ingrediente1': 'pão_de_cima',
              'Ingrediente2': 'presunto',
              'Ingrediente3': 'queijo',
              'Ingrediente4': 'pão_de_baixo'}

dicionario
```

[21]:
```
{'Ingrediente1': 'pão_de_cima',
 'Ingrediente2': 'presunto',
 'Ingrediente3': 'queijo',
 'Ingrediente4': 'pão_de_baixo'}
```

[22]:
```
# As Keys são as Chaves do Dicionario

dicionario.keys()
```

[22]:
```
dict_keys(['Ingrediente1', 'Ingrediente2', 'Ingrediente3', 'Ingrediente4'])
```

[23]:
```
# Os Values são os Valores do Dicionario

dicionario.values()
```

[23]:
```
dict_values(['pão_de_cima', 'presunto', 'queijo', 'pão_de_baixo'])
```

[24]:
```
# Tranformar em Lista invertida para fazer mais um sanduiche?

list( reversed( dicionario.values() ) )
```

[24]:
```
['pão_de_baixo', 'queijo', 'presunto', 'pão_de_cima']
```

Para acessar os dados de um dicionario é um pouco diferente.

[25]:
```
dicionario['Ingrediente4']
```

[25]:
```
'pão_de_baixo'
```

[26]:
```
# Retorna a Chave Ingredinte o Valor dela e remove o Underscore e deixa␣
↪Maiúsculo
dicionario['Ingrediente4'].replace('_', ' ').title()
```

[26]:
```
'Pão De Baixo'
```

Para Utilizar um laço nessas condições é necessário transformar em algum formato de dado de lista, como a lista, tupla ou set, nesse caso ja existe um método que faz isso do proprio dicionario, o items().

Esse método retorna uma tupla com chave e valor que estou transformado para lista para facilitar a visualização

[27]:
```
list(dicionario.items())
```

[27]:
```
[('Ingrediente1', 'pão_de_cima'),
 ('Ingrediente2', 'presunto'),
 ('Ingrediente3', 'queijo'),
 ('Ingrediente4', 'pão_de_baixo')]
```

Agora é pssível **iterar** sobre esses dados.

[28]:
```
for i in list(dicionario.items()):
    print(i)
```

```
('Ingrediente1', 'pão_de_cima')
('Ingrediente2', 'presunto')
('Ingrediente3', 'queijo')
('Ingrediente4', 'pão_de_baixo')
```

[29]:
```
for i in list(dicionario.items()):
    print('Chave: ', i[0])
```

```
Chave:  Ingrediente1
Chave:  Ingrediente2
Chave:  Ingrediente3
Chave:  Ingrediente4
```

[31]:
```
for i in list(dicionario.items()):
    print('Valor: ', i[1])
```

```
Valor:  pão_de_cima
Valor:  presunto
Valor:  queijo
Valor:  pão_de_baixo
```

Mas não é so dessa forma que pode iterar sobre esse tipo de dado, quando você tem uma lista de n dados, é possível utilizar o for e pegar cada um desses índices.

[32]:
```
for i1, i2 in list(dicionario.items()):
    print(f'Chave: {i1} -> Valor: {i2}')
```

```
Chave: Ingrediente1 -> Valor: pão_de_cima
Chave: Ingrediente2 -> Valor: presunto
Chave: Ingrediente3 -> Valor: queijo
Chave: Ingrediente4 -> Valor: pão_de_baixo
```

[33]:
```
# Também não precisa especificar a transformação da Lista

for i1, i2 in dicionario.items():
    print(f'Chave: {i1} -> Valor: {i2}')
```

```
Chave: Ingrediente1 -> Valor: pão_de_cima
Chave: Ingrediente2 -> Valor: presunto
Chave: Ingrediente3 -> Valor: queijo
Chave: Ingrediente4 -> Valor: pão_de_baixo
```

Vamos criar um outro dicionario como exemplo, so que dessa vez com uma lista como valores.

[34]:
```
d2 = {'Lixo': {'Terra', 'Areia', 'Cascalho'},
      'Minerio Normal': {'Ferro', 'Cobre', 'Chumbo', 'Latão'},
      'Minerio Valioso': {'Ouro', 'Prata', 'Platina'},
      'Pedra Preciosa': {'Diamante', 'Ruby', 'Esmeralda', 'Safira'} }

d2
```

[34]:
```
{'Lixo': {'Areia', 'Cascalho', 'Terra'},
 'Minerio Normal': {'Chumbo', 'Cobre', 'Ferro', 'Latão'},
 'Minerio Valioso': {'Ouro', 'Platina', 'Prata'},
 'Pedra Preciosa': {'Diamante', 'Esmeralda', 'Ruby', 'Safira'}}
```

[35]:
```
list(d2.values())
```

[35]:
```
[{'Areia', 'Cascalho', 'Terra'},
 {'Chumbo', 'Cobre', 'Ferro', 'Latão'},
 {'Ouro', 'Platina', 'Prata'},
 {'Diamante', 'Esmeralda', 'Ruby', 'Safira'}]
```

Vamos utilizar um laço para percorrer os dados dessa lista, porém, tem mais uma lista dentro, logo é possível utilizar dois FOR para percorrer cada lista.

```
[36]: # Retornando a Primeira Lista
for chave, valor in d2.items():
    print(valor)
```

```
{'Cascalho', 'Areia', 'Terra'}
{'Chumbo', 'Cobre', 'Ferro', 'Latão'}
{'Prata', 'Platina', 'Ouro'}
{'Diamante', 'Esmeralda', 'Ruby', 'Safira'}
```

```
[37]: for chave, valor in d2.items(): # Primeira Lista
    for indice in valor: # Segunda Lista
        print(indice)
```

```
Cascalho
Areia
Terra
Chumbo
Cobre
Ferro
Latão
Prata
Platina
Ouro
Diamante
Esmeralda
Ruby
Safira
```

```
[38]: print('Lista De Recursos')
print('-_'*20+'\n')

for chave, valor in d2.items():
    print(f'\nItens a Seguir, Correspondem a Seguinte Classificação: -| {chave}␣
↪|-')

    i = 1
    for indice in valor:
        print(f'Item N° {i}: {indice}')
        i += 1
```

```
Lista De Recursos
-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_


Itens a Seguir, Correspondem a Seguinte Classificação: -| Lixo |-
Item N° 1: Cascalho
Item N° 2: Areia
Item N° 3: Terra
```

```
Itens a Seguir, Correspondem a Seguinte Classificação: -| Minerio Normal |-
Item N° 1: Chumbo
Item N° 2: Cobre
Item N° 3: Ferro
Item N° 4: Latão

Itens a Seguir, Correspondem a Seguinte Classificação: -| Minerio Valioso |-
Item N° 1: Prata
Item N° 2: Platina
Item N° 3: Ouro

Itens a Seguir, Correspondem a Seguinte Classificação: -| Pedra Preciosa |-
Item N° 1: Diamante
Item N° 2: Esmeralda
Item N° 3: Ruby
Item N° 4: Safira
```

Podemos Melhorar nossa Mineradora com essas funcionalidades agora! Mas antes, vamos transformar o dicionario em uma lista com todos os Minérios!

5.3 3.3. Um Loop dentro de Listas

Existem formas mais simplificadas de fazer essa transformações anteriores, com as "List comprehension"

[39]:
```
lista_de_itens = []

for k, v in d2.items():
    for i in v:
        lista_de_itens.append( i )

lista_de_itens
```

[39]:
```
['Cascalho',
 'Areia',
 'Terra',
 'Chumbo',
 'Cobre',
 'Ferro',
 'Latão',
 'Prata',
 'Platina',
 'Ouro',
 'Diamante',
 'Esmeralda',
 'Ruby',
 'Safira']
```

[40]:
```
# Para cada Chave, valor nos items do dicionario de minerios, retorna o valor.
[v for k, v in d2.items()]
```

[40]:
```
[{'Areia', 'Cascalho', 'Terra'},
 {'Chumbo', 'Cobre', 'Ferro', 'Latão'},
 {'Ouro', 'Platina', 'Prata'},
 {'Diamante', 'Esmeralda', 'Ruby', 'Safira'}]
```

[41]:
```
list(d2.values())
```

[41]:
```
[{'Areia', 'Cascalho', 'Terra'},
 {'Chumbo', 'Cobre', 'Ferro', 'Latão'},
 {'Ouro', 'Platina', 'Prata'},
 {'Diamante', 'Esmeralda', 'Ruby', 'Safira'}]
```

[42]:
```
# Para indice no set de minerios, mas de onde vem o set?
# O set vem da iteração com a lista de minerios anterior, onde o obejtivo é o␣
 ↪índice

lista_de_minerios = [indice for set_minerios in list(d2.values()) for indice in␣
 ↪set_minerios]
lista_de_minerios
```

[42]:
```
['Cascalho',
 'Areia',
 'Terra',
 'Chumbo',
 'Cobre',
 'Ferro',
 'Latão',
 'Prata',
 'Platina',
 'Ouro',
 'Diamante',
 'Esmeralda',
 'Ruby',
 'Safira']
```

Exemplo dois

[43]:
```
lista = \
[
    ['Olá', 'Prazer'],
    ['Me', 'Chamo'],
    ['Juse', 'Juse2']
]
```

```
# Lista de Listas
lista
```

[43]: [['Olá', 'Prazer'], ['Me', 'Chamo'], ['Juse', 'Juse2']]

[44]:
```
for listinha in lista: # Recupera as listinhas da Listona
    print(listinha)
```

```
['Olá', 'Prazer']
['Me', 'Chamo']
['Juse', 'Juse2']
```

[45]:
```
l = []
for listinha in lista:   # Recupera as Listinhas da Listona
    for reg in listinha:  # Recupera os registros da listinha
        l.append(reg)

l
```

[45]: ['Olá', 'Prazer', 'Me', 'Chamo', 'Juse', 'Juse2']

[46]:
```
                          # Para cada registro na listinha, qual listinha?
# essa listinha aqui man
[reg for listinha in lista for reg in listinha]
# lm for lg       in l4        lm  in  lg
```

[46]: ['Olá', 'Prazer', 'Me', 'Chamo', 'Juse', 'Juse2']

[47]:
```
l4 = [
    [[1, 1], [2, 2]],
    [[3, 3], [4, 4]],
    [[5, 5], [6, 6]]
]

l4
```

[47]: [[[1, 1], [2, 2]], [[3, 3], [4, 4]], [[5, 5], [6, 6]]]

[48]:
```
for linhona in l4:
    print(linhona)
```

```
[[1, 1], [2, 2]]
[[3, 3], [4, 4]]
[[5, 5], [6, 6]]
```

[49]:
```
regs = []
for linhazona in l4:
```

```
    for linhazinha in linhazona:
        for registro in linhazinha:
            regs.append( registro )

regs
```

[49]:
```
[1, 1, 2, 2, 3, 3, 4, 4, 5, 5, 6, 6]
```

[50]:
```
# Sim, não é muito intuitivo, recomendo continuar com o For Classico
[registro for linhazona in l4 for linhazinha in linhazona for registro in␣
 ↪linhazinha]
```

[50]:
```
[1, 1, 2, 2, 3, 3, 4, 4, 5, 5, 6, 6]
```

Para melhorar a Mineradora, vamos fazer alguns ajustes!

Os pesos_dos_minerios nada mais são que a probabilidade da mineradora encontrar algum deses minerios!

[51]:
```
import random

lista_de_minerios  = [indice for set_minerios in list(d2.values()) for indice␣
 ↪in set_minerios]
pesos_dos_minerios = (25, 75, 75, 40, 50, 55, 40, 35, 15, 15, .001, .001, .001,␣
 ↪.01)

# O Método choises faz uma escolha aleatoria de um elemento de uma lista␣
 ↪baseada em pesos
# Os pesos são a probabilidade desse item ser escolhido ou Minerado pela␣
 ↪Mineradora
# O K é a quantidade de vezes que eu quero minerar, por exemplo 2, a mineradora␣
 ↪minerou 2 itens

random.choices( lista_de_minerios, weights=pesos_dos_minerios, k=2)
```

[51]:
```
['Areia', 'Prata']
```

Vamos fazer ela minerar mais 50 vezes.

Vamos fazer as mesmas checagens anteriore s e calcular um lucro apartir de quais minerios formos coletando.

1. Sobre a Quary:
   - Combustíveis apenas Petróleo & Gasolia.
   - O Litro de Gasolina custa 4.50 e o litro de Petróleo custa 4.0.
   - Cada coleta de minerio gasta .03 Litros de Gasolina ou 0.07 Litros de Petróleo.
   - A capacidade da Quary é de 15 Litros.
   - Minera um Minerio a cada 1 segundo.
2. Sobre o Armazenamento de Itens:

- A capacidade de Armazenamento de Itens é 50.
- Itens que são Lixo, podem ser Descartados antes de Chegar no Armazem.
- O Tempo de Armazenamento de UM Item é de 33 segundos.

3. Sobre o Tranporte de Itens:
   - O cano comum de Pedra, tem um bonus de velocidade de 0 e custa 50.
   - O cano de ouro tem um bonus de velocidade de 15 e custa 350.
4. Sobre o descarte de Itens:
   - Cada descarte tem um custo de 0.005 de Energia.
5. Sobre o Processamento de Itens:
   - Cada Item processado em Pó, gasta 30 de Energia e leva 5s. - Po de Chumbo gera: 5 Dinheiro. - Po de Latão gera: 5 Dinheiro. - Po de Cobre gera: 5 Dinheiro. - Po de Ferro gera: 15 Dinheiros. - Po de Prata gera: 45 Money. - Po de Ouro gera: 45 Moneys. - Po de Platina gera: 45 Money. - Cada Preda Preciosa gera 500 Dinheiros.
   - Cada painel Solar gera 10 de energia a cada 1s e custa 150.

Agora trabalhe no processamento de Itens tendo somente: 1000 para Gastar.

[52]:
```
# Setando as Variaveis de Custo & Faturamento
total_money = 1000

po_chumbo = 5
po_latão  = 5
po_cobre = 5
po_ferro = 15
po_prata = 45
po_ouro  = 45
po_plati = 45
pdr_prec = 500

cano_comun = -50
cano_ouro  = -350
l_gasolina = -4.50
l_petroleo = -4.0

# Configuração dos Canos
cano_comun = -50
cano_ouro  = -350
vel_comum = 1
vel_ouro = 15

# Configuração da Quary
quary = True
combustiveis_aceitos = ['Petroleo', 'Gasolina']
minerios_por_seg = 1
capacidade_quary = 15
gasolina_gasta = -.03
petroleo_gasto = -.07
```

```
# Configuração Armazenamento
capacidade = 50
tempo_para_armazenar = 33

# Configuração de Energia
geracao_painel  = 30
segundos_painel = 1
custo_painel    = 150
```

[53]:
```
comb_quary  = 'Petroleo'
lt_na_qary  = 15
armazenados = 0

despesas = abs((lt_na_qary * l_petroleo )) + abs((cano_ouro * 2)) +␣
 ↪abs((cano_comun * 3)) + abs((custo_painel * 3))

# Com 15 Litros a quary pode Minerar por 214 x

if quary:
    if (comb_quary in combustiveis_aceitos) & (lt_na_qary <= capacidade_quary):
        tempo = tempo_para_armazenar - ((vel_ouro * 2) + (vel_comum * 3))

        if (armazenados <= capacidade) & (tempo <= 0):
            painel_solar = 3

            if (painel_solar >= 0) or (despesas > total_money):

                lista_de_minerios  = [indice for set_minerios in list(d2.
 ↪values()) for indice in set_minerios]
                pesos_dos_minerios = (25, 75, 75, 40, 50, 55, 40, 35, 15, 15, .
 ↪001, .001, .001, .01)

                quary_minerios = random.choices( lista_de_itens,␣
 ↪weights=pesos_dos_minerios, k=50)

                for minerio in quary_minerios:

                    if not minerio in lista_de_minerios[:3]:
                            if minerio in lista_de_minerios[-4:]:
                                despesas -= 500

                            elif minerio in lista_de_minerios[4:7]:
                                despesas -= 5

                            elif minerio in lista_de_minerios[7:10]:
                                despesas -= 45
```

```
                            elif minerio == 'Ferro':
                                despesas -= 15
                        else:
                            despesas += .005
                else:
                    print('[ERRO] Você Ultrapassou as Despesas')

            else:
                print(f'[ERRO] Armazen não da conta de armazenar os Itens')

        else:
            print('[ERRO] Verifique a Quary')

else:
    print('[ERRO] Quary esta Desligada!')
```

[54]:
```
print(f'O total de Despesas foi de: {despesas}')
```

```
O total de Despesas foi de: 830.0900000000004
```

Eu sei, esta engraçado haha, mas jaja vamos melhorar ainda mais ;)

6 4.0. Funções & Classes

Funções nada mais é que uma ferramenta sofisticada que evita que o índividuo repita códigos várias vezes, tem um papel muito importante em menater o códgio organizado. Para isso a comunidade do Python desenvolveu várias dessas funções que inclusive fopi utilizado a choises() para fazer a seleção aleatoria dos Minerios que nada mais é que uma função da biblioteca Random.

[55]:
```
# Definição da função com o nome ... que recebe os argumentos ( ... )
def nome_da_funcao( argumento1, argumento2 ):
    print( argumento1 + argumento2 )
```

[56]:
```
# Entre parenteses são onde os argumentos são colocados.
nome_da_funcao( 2, 2 )
```

```
4
```

[57]:
```
# A função recebe 2 Argumentos porém você colocou 3.
nome_da_funcao( 2, 3, 4 )
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Input In [57], in <cell line: 2>()
      1 # A função recebe 2 Argumentos porém você colocou 3.
----> 2 nome_da_funcao( 2, 3, 4 )
```

```
TypeError: nome_da_funcao() takes 2 positional arguments but 3 were given
```

[59]:
```
# É uma boa pratica retornar algum resultado em uma função.
# Mas se ela não retornar nada basta colocar explicitamente o None

def nome_da_funcao( argumento1, argumento2 ):
    print( argumento1 + argumento2 )

    return None
```

[60]:
```
# Definição da Raiz Cubica que recebe um argumento N
# E retorna a raiz Cubica desse Número.

def raiz_cubica( n=2 ):
    return n**(1/3)
```

[61]:
```
# Caso o Usuário não informe um argumento, a função vai executar
# O argumento Default que é 2.

raiz_cubica()
```

[61]:
```
1.2599210498948732
```

[62]:
```
# Com o return você consegue salvar o resultado de uma função em uma variavel

numero = raiz_cubica(5)
numero
```

[62]:
```
1.7099759466766968
```

[63]:
```
[raiz_cubica(i) for i in range(1, 11, 2)]
```

[63]:
```
[1.0,
 1.4422495703074083,
 1.7099759466766968,
 1.912931182772389,
 2.080083823051904]
```

Existem alguns argumentos especiais em funções, como o “args” e o “kwargs”

Que nada mais são que explicitamente mostrar que a função recebe N argumentos.

[64]:
```
# *ARGS
# Quando a sua função recebe n argumentos, geralmente retorna uma tupla.
# Onde cada argumento é um item da tupla.

def aleatorio( *coisas_aleatorias ):
    return [itens for itens in coisas_aleatorias]
```

```
aleatorio(1, 5, "Casa", {'Seila'})
```

[64]:
```
[1, 5, 'Casa', {'Seila'}]
```

Exemplo de onde utilizar, vamos supor que vocÊ quer calcular a média de várias listas e somar as médias no final.

[65]:
```
def mean_sum( *list_ ):
    args = [lista for lista in list_]
    means = []
    for l in args:
        if not type(l) == list:

            # Tenta
            try:
                # Converter o l para o tipo Lista
                l = list( l )
            # Se não conseguir, então gera um Erro
            except ValueError:
                print('Eromeu')

        else:
            mean = sum( l ) / len( l )
            means.append( mean )

    return sum( means )
```

[66]:
```
mean_sum( [1, 2, 3, 45], [1], [9]*5 )
```

[66]:
```
22.75
```

Outra forma de pensar nas funções é quando utiliza-se as bibliotecas do Python.

[67]:
```
# Importando a biblioteca Time
import time
```

[68]:
```
# Objeto da classe time
time.sleep
```

[68]:
```
<function time.sleep>
```

Esse método em especifico faz com que o interpretador espere 5 segundos até executar a próxima linha de código. Muito utilizado em automações.

[69]:
```
# Método da classe time
time.sleep( 5 )
```

[70]:
```
# Esse Método retorna o Timestamp atual
time.time()
```

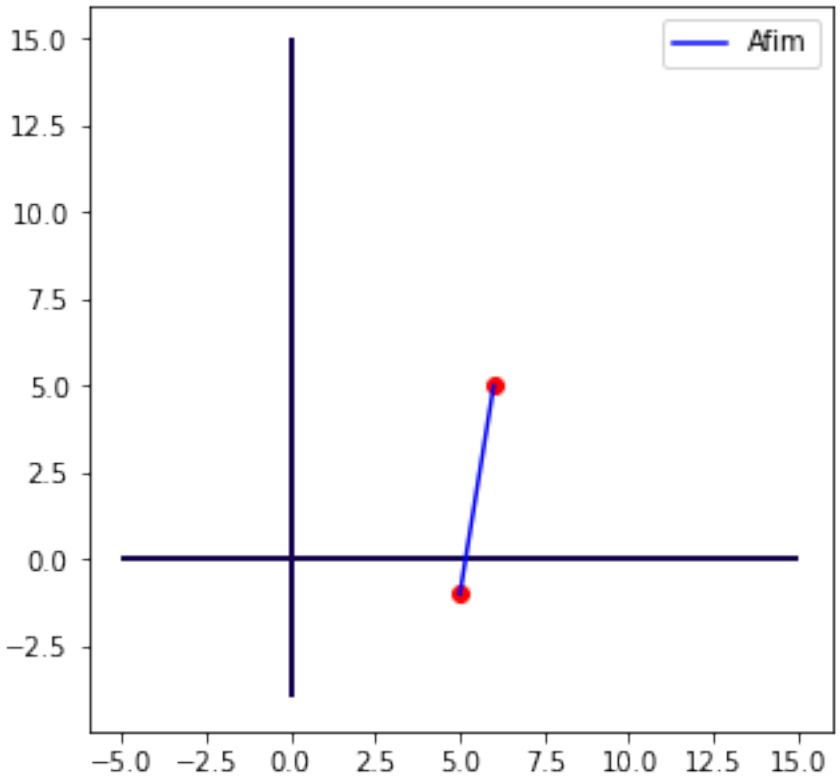
[70]:
```
1646358043.347713
```

[71]:
```
# Importando a biblioteca Math
import math

# Usando a função Raiz Quadrada da Biblioteca Math
int(math.sqrt(25))
```

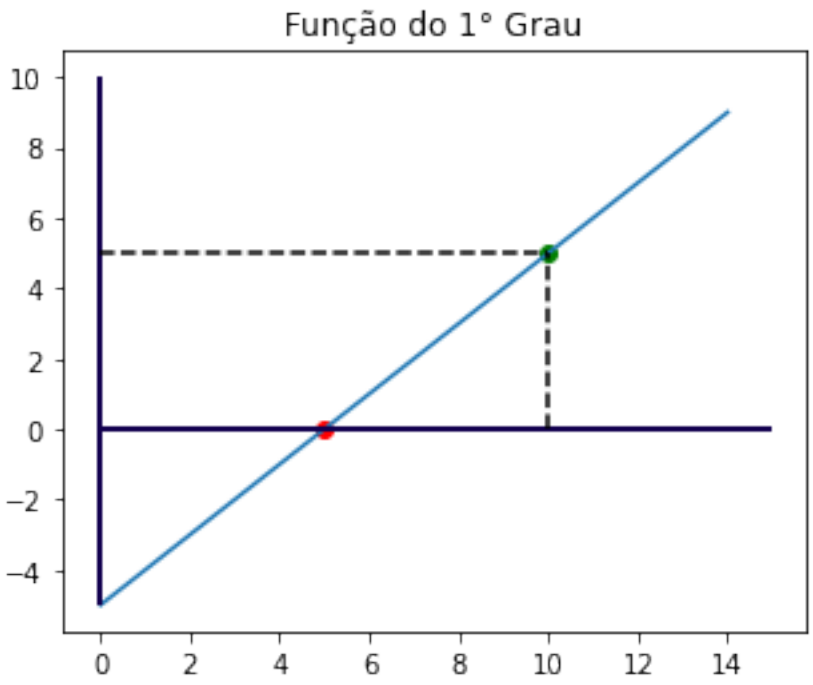
[71]:
```
5
```

Lembra que eu instalei o Matplotlib? Ele é uma biblioteca para desenhar gráficos.

[72]:
```
import matplotlib.pyplot as plt

fig, ax = plt.subplots( figsize=(5, 5) ); # Desenhar o Quadro da Figura
ax.hlines( 0, -5, 15, color="#12004f", linewidth=2, linestyle="-"); # Linha␣
 ↪Horizontal
ax.vlines( 0, -4, 15, color="#12004f", linewidth=2, linestyle="-"); # Linha␣
 ↪Vertical
ax.scatter( [6, 5], [5, -1], color="r" )            # Desenhar as duas bolinhas␣
 ↪nos eixos
ax.plot( [6, 5], [5, -1], c="b", label="Afim" )   # Desenhar a reta ate essas␣
 ↪bolinhas
ax.legend();
```



[73]:
```
fig, ax = plt.subplots( figsize=(5, 4) )
ax.plot( range( -5, 10 ) );
ax.vlines( 0, -5, 10, color="#12004f", linewidth=2, linestyle="-")
ax.hlines( 0, -.1, 15, color="#12004f", linewidth=2, linestyle="-");
ax.vlines( 10, 0, 5, color="k", linestyle="--" )
ax.hlines( 5, 0, 10, color="k", linestyle="--" )
ax.scatter( 5, 0, c="r" )
ax.scatter( 10, 5, c="g" )
ax.set_title("Função do 1° Grau");
```



6.1 4.1. Nossa Quary em Funções

[74]:
```
def run_quary( total_money,
               lista_de_itens,
               lt_na_qary,
               itens_armazenados,
               comb_query       = 'Petroleo',
               qntd_canos_ouro  = 2,
               qntd_canos_comum = 0,
               qntd_painel_solar = 1):

    # Setando as Constantes
    despesas = 0
    l_gasolina = -4.50
    l_petroleo = -4.0

    # Configuração dos Canos
    cano_comun = -50
    cano_ouro  = -350
    vel_comum = 1
    vel_ouro = 15

    # Configuração da Quary
    quary = True
```

```
   combustiveis_aceitos = ['Petroleo', 'Gasolina']
   minerios_por_seg = 1
   capacidade_quary = 15
   gasolina_gasta = -.03
   petroleo_gasto = -.07

   # Configuração Armazenamento
   capacidade = 50
   tempo_para_armazenar = 33

   # Configuração de Energia
   geracao_painel  = 30
   segundos_painel = 1
   custo_painel    = 150

   # Dados que vem de Argumentos
   comb_quary  = comb_query
   lt_na_qary  = lt_na_qary
   armazenados = itens_armazenados
   total_money = total_money
   lista_de_itens = lista_de_itens

   despesas = abs((lt_na_qary * l_petroleo)) + abs((cano_ouro *␣
↪qntd_canos_ouro)) + \
              abs((cano_comun * qntd_canos_comum)) + abs((custo_painel *␣
↪qntd_painel_solar))

   if quary:
       if (comb_quary in combustiveis_aceitos) & (lt_na_qary <=␣
↪capacidade_quary):
           tempo = tempo_para_armazenar - ((vel_ouro * qntd_canos_ouro) +␣
↪(vel_comum * qntd_canos_comum))

           if (armazenados <= capacidade) & (tempo <= 0):
               painel_solar = qntd_painel_solar

               if (painel_solar >= 0) & (despesas < total_money):

                   lista_de_minerios  = [indice for set_minerios in list(d2.
↪values()) for indice in set_minerios]
                   pesos_dos_minerios = (25, 75, 75, 40, 50, 55, 40, 35, 15,␣
↪15, .001, .001, .001, .01)

                   quary_minerios = random.choices( lista_de_itens,␣
↪weights=pesos_dos_minerios, k=50)
```

```
                    for minerio in quary_minerios:

                        if not minerio in lista_de_minerios[:3]:
                                if minerio in lista_de_minerios[-4:]:
                                    despesas -= 500

                                elif minerio in lista_de_minerios[4:7]:
                                    despesas -= 5

                                elif minerio in lista_de_minerios[7:10]:
                                    despesas -= 45

                                elif minerio == 'Ferro':
                                    despesas -= 15
                            else:
                                despesas += .005
                    else:
                        print('[ERRO] Você Ultrapassou as Despesas Iniciais')

                else:
                    print(f'[ERRO] Armazen não da conta de armazenar os Itens')

            else:
                print('[ERRO] Verifique a Quary')

        else:
            print('[ERRO] Quary esta Desligada!')

        return despesas;
```

[75]:
```
run_quary( total_money           = 1500,
           lista_de_itens    = lista_de_itens,
           lt_na_qary        = 15,
           itens_armazenados = 0,
           comb_query        = 'Petroleo',
           qntd_canos_ouro   = 3,
           qntd_canos_comum  = 0,
           qntd_painel_solar = 1 )
```

[75]:
```
820.1150000000008
```

Porem, ainda sim esta muito grande essa função, pode-ser separa em funções menores ainda por exemplo:

[6]:
```
d2 = {'Lixo': {'Areia', 'Cascalho', 'Terra'},
     'Minerio Normal': {'Chumbo', 'Cobre', 'Ferro', 'Latão'},
     'Minerio Valioso': {'Ouro', 'Platina', 'Prata'},
```

```
    'Pedra Preciosa': {'Diamante', 'Esmeralda', 'Ruby', 'Safira'}}
```

[76]:
```
def get_constants():

    # Configurações Gerais
    despesas = 0
    l_gasolina = -4.50
    l_petroleo = -4.0

    # Configuração dos Canos
    cano_comun = -50
    cano_ouro  = -350
    vel_comum = 1
    vel_ouro = 15

    # Configuração da Quary
    quary = True
    combustiveis_aceitos = ['Petroleo', 'Gasolina']
    minerios_por_seg = 1
    capacidade_quary = 15
    gasolina_gasta = -.03
    petroleo_gasto = -.07

    # Configuração Armazenamento
    capacidade = 50
    tempo_para_armazenar = 33

    # Configuração de Energia
    geracao_painel  = 30
    segundos_painel = 1
    custo_painel    = 150

    results = {
        'Geral': [despesas, l_gasolina, l_petroleo],
        'Canos': [cano_comun, cano_ouro, vel_comum, vel_ouro],
        'Quary': [quary, combustiveis_aceitos, minerios_por_seg,␣
 ↪capacidade_quary, gasolina_gasta, petroleo_gasto],
        'Armazem': [capacidade, tempo_para_armazenar],
        'Energia': [geracao_painel, segundos_painel, custo_painel]
    }

    return results

def run_quary( total_money,
               lista_de_itens,
               lt_na_qary,
               itens_armazenados,
```

```
              comb_query       = 'Petroleo',
              qntd_canos_ouro  = 2,
              qntd_canos_comum = 0,
              qntd_painel_solar = 1):

   c = get_constants()

   # Dados que vem de Argumentos
   comb_quary  = comb_query
   lt_na_qary  = lt_na_qary
   armazenados = itens_armazenados
   total_money = total_money
   lista_de_itens = lista_de_itens

   c['Geral'][0] = abs((lt_na_qary * c['Geral'][2])) + abs((c['Canos'][1] *␣
↪qntd_canos_ouro)) + \
                   abs((c['Canos'][0] * qntd_canos_comum)) +␣
↪abs((c['Energia'][-1] * qntd_painel_solar))

   if quary:
       if (comb_quary in combustiveis_aceitos) & (lt_na_qary <=␣
↪capacidade_quary):
           tempo = c['Armazem'][-1] - ((c['Canos'][-1] * qntd_canos_ouro) +␣
↪(c['Canos'][2] * qntd_canos_comum))

           if (armazenados <= c['Armazem'][0]) & (tempo <= 0):

               if (qntd_painel_solar >= 0) & (c['Geral'][0] < total_money):

                   lista_de_minerios  = [indice for set_minerios in list(d2.
↪values()) for indice in set_minerios]
                   pesos_dos_minerios = (25, 75, 75, 40, 50, 55, 40, 35, 15,␣
↪15, .001, .001, .001, .01)

                   quary_minerios = random.choices( lista_de_itens,␣
↪weights=pesos_dos_minerios, k=50)

                   for minerio in quary_minerios:

                       if not minerio in lista_de_minerios[:3]:
                               if minerio in lista_de_minerios[-4:]:
                                   c['Geral'][0] -= 500

                               elif minerio in lista_de_minerios[4:7]:
                                   c['Geral'][0] -= 5
```

```
                                elif minerio in lista_de_minerios[7:10]:
                                    c['Geral'][0] -= 45

                                elif minerio == 'Ferro':
                                    c['Geral'][0] -= 15
                            else:
                                c['Geral'][0] += .005
                    else:
                        print('[ERRO] Você Ultrapassou as Despesas Iniciais')

                else:
                    print(f'[ERRO] Armazen não da conta de armazenar os Itens')

            else:
                print('[ERRO] Verifique a Quary')

        else:
            print('[ERRO] Quary esta Desligada!')

        return c['Geral'][0];
```

[814]:
```
run_quary( total_money           = 1500,
           lista_de_itens    = lista_de_itens,
           lt_na_qary        = 15,
           itens_armazenados = 0,
           comb_query        = 'Petroleo',
           qntd_canos_ouro   = 3,
           qntd_canos_comum  = 0,
           qntd_painel_solar = 1 )
```

[814]:
```
695.0950000000009
```

Enfim vocês ja entenderam a ideia, agora é so separar mais e mais :)

[77]:
```
def get_constants():

    # Configurações Gerais
    despesas = 0
    l_gasolina = -4.50
    l_petroleo = -4.0

    # Configuração dos Canos
    cano_comun = -50
    cano_ouro  = -350
    vel_comum = 1
    vel_ouro = 15
```

```
    # Configuração da Quary
    quary = True
    combustiveis_aceitos = ['Petroleo', 'Gasolina']
    minerios_por_seg = 1
    capacidade_quary = 15
    gasolina_gasta = -.03
    petroleo_gasto = -.07

    # Configuração Armazenamento
    capacidade = 50
    tempo_para_armazenar = 33

    # Configuração de Energia
    geracao_painel  = 30
    segundos_painel = 1
    custo_painel    = 150

    results = {
        'Geral': [despesas, l_gasolina, l_petroleo],
        'Canos': [cano_comun, cano_ouro, vel_comum, vel_ouro],
        'Quary': [quary, combustiveis_aceitos, minerios_por_seg,␣
↪capacidade_quary, gasolina_gasta, petroleo_gasto],
        'Armazem': [capacidade, tempo_para_armazenar],
        'Energia': [geracao_painel, segundos_painel, custo_painel]
    }

    return results

def run_quary( comb_quary, lt_na_qary, armazenados, total_money, lista_de_itens,
               qntd_canos_ouro, qntd_canos_comum, qntd_painel_solar ):

    c = get_constants()

    c['Geral'][0] = abs((lt_na_qary * c['Geral'][2])) + abs((c['Canos'][1] *␣
↪qntd_canos_ouro)) + \
                    abs((c['Canos'][0] * qntd_canos_comum)) +␣
↪abs((c['Energia'][-1] * qntd_painel_solar))

    if quary:
        if (comb_quary in combustiveis_aceitos) & (lt_na_qary <=␣
↪capacidade_quary):
            tempo = c['Armazem'][-1] - ((c['Canos'][-1] * qntd_canos_ouro) +␣
↪(c['Canos'][2] * qntd_canos_comum))

            if (armazenados <= c['Armazem'][0]) & (tempo <= 0):

                if (qntd_painel_solar >= 0) & (c['Geral'][0] < total_money):
```

```
                    lista_de_minerios  = [indice for set_minerios in list(d2.
↪values()) for indice in set_minerios]
                    pesos_dos_minerios = (25, 75, 75, 40, 50, 55, 40, 35, 15,␣
↪15, .001, .001, .001, .01)

                    quary_minerios = random.choices( lista_de_itens,␣
↪weights=pesos_dos_minerios, k=50)

                    for minerio in quary_minerios:

                        if not minerio in lista_de_minerios[:3]:
                                if minerio in lista_de_minerios[-4:]:
                                    c['Geral'][0] -= 500

                                elif minerio in lista_de_minerios[4:7]:
                                    c['Geral'][0] -= 5

                                elif minerio in lista_de_minerios[7:10]:
                                    c['Geral'][0] -= 45

                                elif minerio == 'Ferro':
                                    c['Geral'][0] -= 15
                            else:
                                c['Geral'][0] += .005
                    else:
                        print('[ERRO] Você Ultrapassou as Despesas Iniciais')

                else:
                    print(f'[ERRO] Armazen não da conta de armazenar os Itens')

            else:
                print('[ERRO] Verifique a Quary')

        else:
            print('[ERRO] Quary esta Desligada!')

        return c['Geral'][0];

def set_up_quary( total_money,
                  lista_de_itens,
                  lt_na_qary,
                  itens_armazenados,
                  comb_query       = 'Petroleo',
                  qntd_canos_ouro  = 2,
                  qntd_canos_comum = 0,
                  qntd_painel_solar = 1):
```

```
    comb_quary  = comb_query
    lt_na_qary  = lt_na_qary
    armazenados = itens_armazenados
    total_money = total_money
    lista_de_itens = lista_de_itens
    qntd_canos_ouro   = qntd_canos_ouro
    qntd_canos_comum  = qntd_canos_comum
    qntd_painel_solar = qntd_painel_solar

    quary_results = run_quary( comb_quary  = comb_query,
                               lt_na_qary  = lt_na_qary,
                               armazenados = itens_armazenados,
                               total_money = total_money,
                               lista_de_itens = lista_de_itens,
                               qntd_canos_ouro   = qntd_canos_ouro,
                               qntd_canos_comum  = qntd_canos_comum,
                               qntd_painel_solar = qntd_painel_solar)

    return quary_results;
```

[78]:
```
set_up_quary( total_money        = 1500,
              lista_de_itens     = lista_de_itens,
              lt_na_qary         = 15,
              itens_armazenados = 0,
              comb_query         = 'Petroleo',
              qntd_canos_ouro    = 3,
              qntd_canos_comum   = 0,
              qntd_painel_solar = 1 )
```

[78]:
```
735.1000000000008
```

Chega, cansei, ainda dava de melhorar, mas vamos supor que você ja separou bastante o seu códgido :D

6.2 4.2. A Classe da Quary

As classes nada mais são que o proximo passo de um script mais profissional. Geralmente ela é Criada com a palavra reservada Class e em CamelCase.

Uma dica muito importante antes de criar uma classe é ja ter definido todas as funçoes, pois é muito facil perder as informações e no final nem as funções funcionam mais haha.

Não irei entrar em muitos detalhes sobre as funções construtoras e super classes etc.

[7]:
```
import random

class RunQuary():
```

```
    '''
    # Aqui vai a documentação de sua Classe
    '''

    def __init__( self ): # O init serve basicamente para deixar preparado␣
↪algumas variáveis
        self.d2 = {'Lixo': {'Areia', 'Cascalho', 'Terra'},
                   'Minerio Normal': {'Chumbo', 'Cobre', 'Ferro', 'Latão'},
                   'Minerio Valioso': {'Ouro', 'Platina', 'Prata'},
                   'Pedra Preciosa': {'Diamante', 'Esmeralda', 'Ruby',␣
↪'Safira'}}

        self.lista_de_minerios  = [indice for set_minerios in list(self.d2.
↪values()) for indice in set_minerios]

    def get_constants( self ): # Ponteiro Obrigatório

        # Configurações Gerais
        despesas = 0
        l_gasolina = -4.50
        l_petroleo = -4.0

        # Configuração dos Canos
        cano_comun = -50
        cano_ouro  = -350
        vel_comum = 1
        vel_ouro = 15

        # Configuração da Quary
        quary = True
        combustiveis_aceitos = ['Petroleo', 'Gasolina']
        minerios_por_seg = 1
        capacidade_quary = 15
        gasolina_gasta = -.03
        petroleo_gasto = -.07

        # Configuração Armazenamento
        capacidade = 50
        tempo_para_armazenar = 33

        # Configuração de Energia
        geracao_painel  = 30
        segundos_painel = 1
        custo_painel    = 150

        results = {
            'Geral': [despesas, l_gasolina, l_petroleo],
```

```
            'Canos': [cano_comun, cano_ouro, vel_comum, vel_ouro],
            'Quary': [quary, combustiveis_aceitos, minerios_por_seg,␣
↪capacidade_quary, gasolina_gasta, petroleo_gasto],
            'Armazem': [capacidade, tempo_para_armazenar],
            'Energia': [geracao_painel, segundos_painel, custo_painel]
        }

        return results
                    # Again
    def run_quary( self, comb_quary, lt_na_qary, armazenados, total_money,␣
↪#lista_de_itens,
                      qntd_canos_ouro, qntd_canos_comum, qntd_painel_solar ):
            # Aqui tmb
        c = self.get_constants()

        c['Geral'][0] = abs((lt_na_qary * c['Geral'][2])) + abs((c['Canos'][1]␣
↪* qntd_canos_ouro)) + \
                        abs((c['Canos'][0] * qntd_canos_comum)) +␣
↪abs((c['Energia'][-1] * qntd_painel_solar))

        if quary:
            if (comb_quary in c['Quary'][1]) & (lt_na_qary <= c['Quary'][3]):
                tempo = c['Armazem'][-1] - ((c['Canos'][-1] * qntd_canos_ouro)␣
↪+ (c['Canos'][2] * qntd_canos_comum))

                if (armazenados <= c['Armazem'][0]) & (tempo <= 0):

                    if (qntd_painel_solar >= 0) & (c['Geral'][0] < total_money):
                                                                        #␣
↪Self Aqui também
                        #lista_de_minerios  = [indice for set_minerios in␣
↪list(self.d2.values()) for indice in set_minerios]
                        pesos_dos_minerios = (25, 75, 75, 40, 50, 55, 40, 35,␣
↪15, 15, .001, .001, .001, .01)

                        quary_minerios = random.choices( self.
↪lista_de_minerios, weights=pesos_dos_minerios, k=50)

                        for minerio in quary_minerios:

                            if not minerio in self.lista_de_minerios[:3]:
                                    if minerio in self.lista_de_minerios[-4:]:
                                        c['Geral'][0] -= 500

                                    elif minerio in self.lista_de_minerios[4:7]:
                                        c['Geral'][0] -= 5
```

```
                                    elif minerio in self.lista_de_minerios[7:
↪10]:
                                        c['Geral'][0] -= 45

                                    elif minerio == 'Ferro':
                                        c['Geral'][0] -= 15
                                else:
                                    c['Geral'][0] += .005
                        else:
                            print('[ERRO] Você Ultrapassou as Despesas Iniciais')

                    else:
                        print(f'[ERRO] Armazen não da conta de armazenar os Itens')

                else:
                    print('[ERRO] Verifique a Quary')

            else:
                print('[ERRO] Quary esta Desligada!')

            return c['Geral'][0];

    def set_up_quary( self,
                      total_money,
                      #lista_de_itens,
                      lt_na_qary,
                      itens_armazenados,
                      comb_query       = 'Petroleo',
                      qntd_canos_ouro  = 2,
                      qntd_canos_comum = 0,
                      qntd_painel_solar = 1):

            quary_results = self.run_quary( comb_query,
                                            lt_na_qary,
                                            itens_armazenados,
                                            total_money,
                                            #lista_de_itens,
                                            qntd_canos_ouro,
                                            qntd_canos_comum,
                                            qntd_painel_solar)

            return quary_results;
```

[14]:
```
# Instanciando a Classe Quary
quary = RunQuary()
```

[3]:
```
# Dicionario Fixo do construtor
quary.d2
```

[3]:
```
{'Lixo': {'Areia', 'Cascalho', 'Terra'},
 'Minerio Normal': {'Chumbo', 'Cobre', 'Ferro', 'Latão'},
 'Minerio Valioso': {'Ouro', 'Platina', 'Prata'},
 'Pedra Preciosa': {'Diamante', 'Esmeralda', 'Ruby', 'Safira'}}
```

[9]:
```
# Pipeline para iniciar a Quary

if __name__ == '__main__':
    quary = RunQuary()

    result = quary.set_up_quary( total_money       = 1500,
                                 #lista_de_itens    = lista_de_minerios,
                                 lt_na_qary        = 15,
                                 itens_armazenados = 0,
                                 comb_query        = 'Petroleo',
                                 qntd_canos_ouro   = 3,
                                 qntd_canos_comum  = 0,
                                 qntd_painel_solar = 1 )

    print( 'Despesas: ', result )
```

```
Despesas:  970.1200000000026
```

Acabou ; - ;