



Uma introdução à
Localização (l10n) e
Internacionalização (i18n)

Live de Python #290

Enquanto eu preparava este material, entrei na toca do coelho. Achei que era melhor termos duas lives boas do que uma única corrida!



Aviso



Enquanto eu preparava este material, entrei na toca do coelho. Achei que era melhor termos duas lives boas do que uma única corrida!

N	Assunto	Data
290	Teoria e internacionalização de mensagens (gettext)	28/07
292	Localização (locale) e bibliotecas complementares	11/08



Aviso





1. Os princípios da coisa

Um pouco de teoria do i18n e l10n

2. gettext

Criando arquivos de tradução

3. Um exemplo prático

Construindo uma CLI localizado (p1)

4. Algumas coisas mais

"Continuous Localization", plataformas, softwares, pacotes e etc...



apoia.se/livedepython



pix.dunossauro@gmail.com



patreon.com/dunossauro



Ajude o projeto <3



Albano Maywitz, Alexandre Costa, Alexandre Harano, Alexandre Lima, Alexandre Takahashi, Alexandre Villares, Alfredo Braga, Alfredomoraes, Alfredo Neto, Alysson Oliveira, Andre, Andre Makoski, André Oliveira, Andre Paula, Antonio Filho, Apc 16, Apolo Ferreira, Arthur Santiago, Augusst0o, Augusto Domingos, Aurelio Costa, Bed, Belisa Arnhold, Bernardo Fontes, Bernarducs, Biancarosa, Brisa Nascimento, Bruno Batista, Bruno Bereoff, Bruno Freitas, Bruno Ramos, Bruno Russian, Bug_elseif, Canibasami, Carlos Gonçalves, Carlos Henrique, Cauã Oliveira, Celio Araujo, Christian Fischer, Claudemir Cruz, Cleiton Fonseca, Controlado, Curtos Treino, Daniel Aguiar, Daniel Brito, Daniel Bruno, Daniel Souza, Daniel Wojcickoski, Danilo Boas, Danilo Silva, Darcioalberico_sp, David Couto, Dh44s, Diego Guimarães, Dilan Nery, Dunossauro, Edgar, Elias Soares, Emerson Rafael, Érico Andrei, Everton Silva, Ewertonbello, Fabio Barros, Fábio Belotto, Fabio Faria, Fabiokleis, Felipe Adeildo, Felipe Augusto, Felipe Corrêa, Fernanda Prado, Fernandocelmer, Ferrabras, Fichele Marias, Fightorfall, Francisco Aclima, Franklin Sousa, Frederico Damian, Fulvio Murenu, Gabriel Lira, Gabriel Mizuno, Gabriel Paiva, Gabriel Simonetto, Gardienbr, Geilton Cruz, Geisler Dias, Giuliano Silva, Glauber Duma, Gnomo Nimp, Grinaode, Guibeira, Guilherme Felitti, Guilherme Ostrock, Gustavo Suto, Harold Gautschi, Heitor Fernandes, Hellyson Ferreira, Helton, Helvio Rezende, Henri Alves, Henrique Andrade, Henrique Machado, Henriquesebastiao, Herian Cavalcante, Hiago Couto, Idlelive, Ivan Santiago, Ivansantiagojr, Janael Pinheiro, Jean Victor, Jeferson Vitalino, Jefferson Antunes, Jerry Ubiratan, Jhonata Medeiros, Jhon Gonçalves, Joaocuginotto, João Pena, Joao Rocha, Joarez Wernke, Jonas Araujo, Jonatas Leon, Jonatas_silva11, Jose Barroso, Joseíto Júnior, José Predo), Josir Gomes, Jota_lugs, Jplay, Jrborba, Ju14x, Juan Felipe, Juli4xpy, Juliana Machado, Julio Franco, Julio Silva, Kacoplay, Kaio Peixoto, Kakaroto, Kalevi, Knaka, Krisquee, Lara Nápoli, Leandro O., Leandro Pina, Leandro Vieira, Leonardo Mello, Leonardo Nazareth, Lilian Pires, Lisandro Pires, Lucas Carderelli, Lucas Castro, Lucasgcode, Lucas Mello, Lucas Mendes, Lucas Nascimento, Lucas Polo, Lucas Schneider, Luciano_ratamero, Luciano Ratamero, Lúcia Silva, Luidduarte, Luis Ottoni, Luiz Duarte, Luiz Martins, Luiz Paula, Luiz Perciliano, Marcelo Araujo, Marcelo Fonseca, Marcio Freitas, Marcos Almeida, Marcos Oliveira, Maria Santos, Marina Passos, Mateusamorim96, Mateus Ribeiro, Matheus Mendez, Matheus Vian, Medalutadorespacialx, Mlevi Lsantos, Murilo Carvalho, Nhambu, Omatheusfc, Oopaze, Ostuff_fps, Otávio Carneiro, Patrick Felipe, Pytonyc, Rafael Ferreira, Rafael Fontenelle, Rafael Lopes, Rafael Romão, Raimundo Ramos, Ramayana Menezes, Renan, Renan Sebastião, Renato José, Rene Pessoto, Renne Rocha, Ricardo Silva, Ricardo Viana, Ric_fv, Richard Sousa, Rinaldo Magalhaes, Robsonpiere, Rodrigo Barretos, Rodrigo Santana, Rodrigo Vieira, Rogério Nogueira, Rui Jr, Samael Picoli, Samanta Cicilia, Santhiago Cristiano, Scrimf00x, Sherlock Holmes, Shinodinho, Shirakawa, Tarcisio Miranda, Tenorio, Téó Calvo, Teomewhy, Tharles Andrade, Thiago, Thiago Araujo, Thiago Lucca, Thiago Paiva, Tiago, Tomás Tamantini, Trojanxds, Valdir, Varlei Menconi, Viniciusfk9, Vinicius Silva, Vinicius Souza, Vinicius Stein, Vladimir Lemos, Williamslews, Willian Lopes, Will_sca, Xxxxxxxxxx, Zero! Studio



Obrigado você



Os princípios por
trás.

Intro
dução

É de comer?



Um dos pontos principais do desenvolvimento é ter um software com uma base maior de pessoas usando. Isso tanto de forma comercial, quanto de uma visão de comunidade.

Uma das formas de fazer isso é **transcender barreiras geográficas e de idiomas**.

Oferecendo o acesso a públicos que falam línguas diferentes, têm padrões regionais diferentes e culturas diferentes.

O software precisa ser **localizado** para regiões diferentes.

Localização – I10n



Localização é o nome dado ao processo de adaptar um software para atender às necessidades linguísticas, culturais e regionais de um público específico. Em uma definição mais formal, segundo a LISA (Localization Industry Standards Association) (Esselink, 2000):

"A localização envolve pegar um produto e torná-lo linguisticamente e culturalmente apropriado para o local de destino (país/região e idioma) onde ele será usado e vendido."

Localização – I10n



No primeiro momento, podemos pensar especificamente em **tradução**. Traduzir nosso software para que a língua escolhida por quem o usa. Embora esse seja o passo mais óbvio dos necessários, muitas outras coisas devem ser consideradas.

Os "**padrões regionais**" que devem ser levados em conta. Como:

- **Medidas**
- **Datas**
- **Moeda**
- **Telefones**
- **Ordenação:**
- E muito mais...

"Padrões" regionais



- **Medidas:** Diferentes locais usam sistemas de unidades diferentes
 - Sistema métrico (metros, litros, quilos)
 - Sistema imperial (pés, galões, libras)
- **Datas:** Os formatos de datas variam dependendo da região
 - No Brasil, usamos dia/mês/ano
 - China: ano/mês/dia.
- **Moeda:** A moeda varia conforme a região.
 - Real no Brasil
 - Euro na Europa
 - Dólar nos Estados Unidos
- Telefones
- Ordenação
- E muito mais...

"Padrões" regionais



- Medidas
- Datas
- Moeda
- **Telefones:** Em localidades diferentes, os telefones têm formatos diferentes.
 - Na Índia, por exemplo, temos códigos de área de quatro dígitos
- **Ordenação:** Dependendo do alfabeto usado, as regras de ordenação são diferentes.
- **E muito mais...**
 - **Direção do texto**
 - **Linhas ou colunas**
 - **Charset**
 - ...

"Quando um software se adapta a essas questões regionais, além da tradução do idioma, dizemos que o programa é localizado para a região X."



Localização - I10n



Internacionalização – i18n



Enquanto a **localização** é o ato concreto de regionalizar um sistema, a **internacionalização** é o processo de construção do software para que ele possa ser flexível a diferentes idiomas e regiões.

A internacionalização envolve o **desenvolvimento de código com a flexibilidade necessária para ser usado globalmente**, independentemente de sua origem territorial.

Internacionalização – i18n



Na definição do LISA (Esselink, 2000):

*"Internacionalização é o processo de **generalizar um produto** para que ele possa lidar com múltiplos idiomas e convenções culturais **sem a necessidade de redesign**. A internacionalização ocorre no nível do design do programa e do desenvolvimento de documentos."*

Internacionalização – i18n



A **internacionalização** deve ser realizada **antes** da **localização**.

Isso significa que, durante a criação do software, ele precisa ser projetado de forma que possa ser facilmente adaptado a novos idiomas e regiões, sem exigir grandes mudanças no código.

- mensagens (strings) precisam ser extraídas sem alterar o comportamento do programa
- os números usados pela aplicação precisam ser formatados conforme a região.
- ...

É a parte da engenharia de software para a construção de um codebase adaptado para a localização ser possível.

Termos



Como os termos foram criados pela indústria, é bastante complicado precisar quando eles apareceram pela primeira vez.

Segundo **i18nguy** em *Origin Of The Abbreviation I18n*, a origem dos termos é desconhecida. Ele suspeita que ambos os termos tenham surgido na década de 80:

"The term "internationalization" was used as far back as 1985 at Apple and DEC. Unix seems to have picked it up at least in the late 80s. "

Numerônimos



Mas e as abreviações? i18n, l10n?

"O termo **numerônimo** se refere a palavras encurtadas com o uso de números para representar partes da palavra. No caso de i18n, o número 18 corresponde ao número de letras entre o "i" e o "n" na palavra "internationalization". Usamos numerônimos também em termos modernos como: **k8s** (Kubernetes), **o11y** (Observability), etc."

Um mundo sem padrões



Embora a internacionalização e a localização sejam essenciais para atingir públicos globais, **não existe um padrão universal** adotado por todas as plataformas.

Cada ecossistema adota sua própria abordagem, o que pode gerar confusão, principalmente durante a fase de aprendizado. Levando em conta somente as traduções como exemplo:

- **Linux:** Usa o padrão **gettext** do projeto GNU
- **Windows:** Arquivos **.resx**, que armazenam recursos como textos traduzidos e outros dados específicos de localização.
- **MacOS:** Arquivos **.strings**, que contêm as traduções necessárias
- **Web:** especialmente com **JavaScript**, bibliotecas como **i18next** são populares, utilizando arquivos de tradução em JSON ou YAML

Prática

Um projetinho

Assunto deveras abstrato



Então vamos usar uma aplicaçãozinha que fiz especialmente para essa live.
Sempre que aprendermos um conceito, vamos aplicar nesse projeto.

projeto de moedas

Biblio tecas

Nativas
(gettext, locale)

Bibliotecas padrão

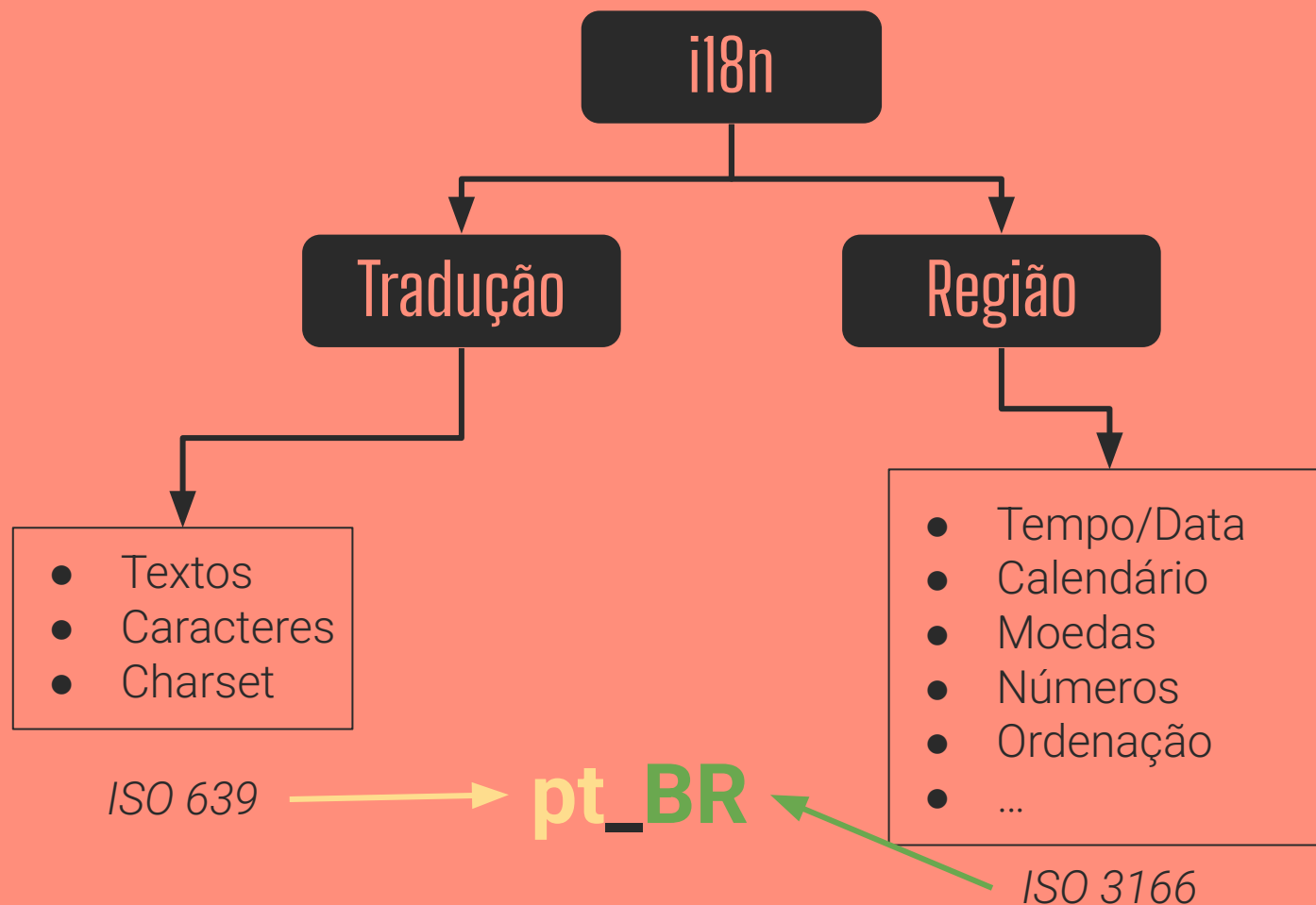


As bibliotecas padrões para fazer i18n são influenciadas por dois padrões:

- **GNU/gettext**: para arquivos de tradução
- **POSIX**: especificação para o **Locale**

Embora a relação entre esses dois padrões trabalhando em conjunto seja um pouco difícil de traçar historicamente, eles são bem documentados em conjunto na **OpenI18N**, criada pelo hoje extinto **Free Standards Group**. Essa iniciativa foi incorporada a **Linux Standard Base** (LSB)

Um exemplo visual





A **gettext** trabalha em conjunto com o ecossistema do GNU/gettext, uma caixa de ferramentas e padrões para desenhados para minimizar o impacto da internacionalização. A ideia principal dessa ferramenta é isolar as strings do código para tradução.

```
# exemplo_00.py
import gettext

_ = gettext.gettext

print(_('Hello'))
```

Olá!



Como todo início de estudo requer um "olá mundo" para não azarar o aprendizado, começaremos por ele:

```
# exemplo_00.py
import gettext

_ = gettext.gettext

print(_('Hello'))
```

O `_` como alias para **gettext.gettext** é um padrão definido para as strings que devam ser internacionalizadas possam ser encontradas pelo "**raspador**" dos códigos-fonte.



Agora que temos uma string para ser internacionalizada, podemos conhecer as ferramentas de terminal do GNU/gettext.

- **xgettext**: "Raspa" as strings i18n do texto
- **msginit**: Inicia um arquivo de traduções em um idioma específico
- **msgmerge**: Atualiza uma tradução, adiciona novas mensagens
- **msgfmt**: "Compila" o arquivo com as traduções
- **msgunfmt**: "Decompila" o arquivo de compilado

> Caso esteja usando windows, o gettext está disponível para a plataforma no repositório: <https://github.com/mlocati/gettext-iconv-windows>

Um glossário básico – GNU/gettext



- **xgettext:** "Raspa" as strings i18n do texto e
 - Cria um template de tradução (**.pot** [Portable Object Template])
- **msginit:** Inicia um arquivo de traduções em um idioma específico
 - Cria um arquivo de tradução (**.po** [Portable object])
- **msgmerge:** Atualiza uma tradução, adiciona novas mensagens
 - Adiciona as novas mensagens do template ao **po** da lingua
- **msgfmt:** "Compila" o arquivo com as traduções
 - Em um arquivo **.mo** [Machine object]
- **msgunfmt:** "Decompila" o arquivo de compilado
 - **.mo -> .po**

xgettext



Vamos começar com o **xgettext**.

Ele recebe um arquivo ou uma lista de arquivos .py e encontra todas as mensagens com o padrão ``_('string')`` ou `gettext('string')` e gera um template de tradução:

```
xgettext -o messages.pot exemplo_00.py
# Uma lista de arquivos .py pode ser enviada
# Se precisar usar com diretórios pode usar o `find` em conjunto
# algo como:
# find . -name '*.py' -print0 | xargs -0 xgettext -o messages.pot
```

O arquivo .pot



Após alguns diversos metadados (que veremos a seguir), o arquivo tem basicamente um **msgid**, o identificador da mensagem. E o **msgstr**, que é onde a tradução será encaixada.

```
#: exemplo_00.py:5
msgid "Hello"
msgstr ""
```

Os metadados



```
Project-Id-Version: PACKAGE VERSION
Report-Msgid-Bugs-To:
POT-Creation-Date: 2025-07-21 08:06-0300
PO-Revision-Date: YEAR-MO-DA HO:MI+ZONE
Last-Translator: FULL NAME <EMAIL@ADDRESS>
Language-Team: LANGUAGE <LL@li.org>
Language:
MIME-Version: 1.0
Content-Type: text/plain; charset=CHARSET
Content-Transfer-Encoding: 8bit
Plural-Forms: nplurals=INTEGER; plural=EXPRESSION;
```



Agora que temos um arquivo **.pot**, podemos usar ele como base para a criação de um arquivo de tradução para uma língua específica.

Como, por exemplo, 'pt_BR', resultando em um arquivo **.po**

```
msginit --locale=pt_BR.UTF-8 \  
        --input=messages.pot \  
        --output=pt_BR.po
```

Como a flag **-locale** deve ser criada?

ISO 639 _ ISO 3166-1



A construção do locale é dada por duas ISOs específicas. Que delimitam o código da língua (639 set 1) e o código de país (3166-1, alpha 2).

ISO 639 → **pt**—**BR** ← ISO 3166

** Nota especial para línguas não associadas a países. Elas não levam a segunda parte. Casos de línguas artificiais, com Esperanto, somente eo.*

charset



Note que o charset **UTF-8** também é passado. O charset é extremamente importante, pois ele quem define como os caracteres serão apresentados.

Charset é um oceano a parte, que não focarei nesse texto, pois ele claramente merece uma aula específica sobre ele e sua relação com unicode.

```
msginit --locale=pt_BR.UTF-8 \  
        --input=messages.pot \  
        --output=pt_BR.po
```

> Sempre que possível, use UTF-8 nos arquivos `.po` para evitar problemas de codificação ao lidar com idiomas não-latinos.

Montando a tradução



Agora podemos preencher os **msgstr** e também olhar as alterações no cabeçalho.

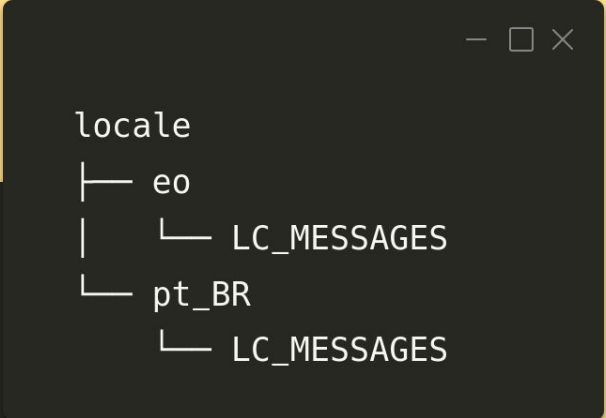
```
Language: pt_BR
Content-Type: text/plain; charset=UTF-8
Plural-Forms: nplurals=2; plural=(n > 1);

#: exemplo_00.py:3
msgid "Hello"
msgstr "Olá"
```

Agora, com a tradução pronta, podemos gerar o arquivo final de tradução com **msgfmt**.

Uma observação especial é que existe uma hierarquia de diretórios que devem ser respeitados.

```
msgfmt pt_BR.po \  
-o locale/pt_BR/LC_MESSAGES/messages.mo
```



```
locale  
├── eo  
│   └── LC_MESSAGES  
└── pt_BR  
    └── LC_MESSAGES
```

De volta ao código!



Agora que temos o **msgid** traduzido e no local correto, precisamos adaptar nosso código para lidar com as traduções:

```
import gettext

t = gettext.translation('messages', localedir='locale')

t.install()

print(_('Hello'))
```

A função **gettext.translation** vai buscar pelas mensagens em **locale** no contexto de **messages**.

De volta ao código!



Agora que temos o **msgid** traduzido e no local correto, precisamos adaptar nosso código para lidar com as traduções:

```
import gettext
```

```
t = gettext.translation('messages', localedir='locale')
```

```
t.install()
```

```
print(_('Hello'))
```

Contexto (nome do .po)

A função **gettext.translation** vai buscar pelas mensagens em **locale** no contexto de **messages**.

O grande momento



Contudo, a variável de ambiente **LANG** será considerada na hora de executar o código, traduzindo o contexto de mensagens:

```
python exemplo_00.py
```

```
Hello
```

```
# $env:LANG = "pt_BR" - Para windows via powershell
```

```
LANG='pt_BR' python exemplo_00.py
```

```
Olá
```

Com isso, temos um "Olá mundo" localizado em português do Brasil <3

Beleza... As coisas às vezes dão errado



Se eu não tiver o idioma de LANG nas mensagens, vamos ter um raise

```
LANG='es_AR' python exemplo_00.py
Traceback (most recent call last):
  # ...
  File "/usr/lib/python3.13/gettext.py", line 537, in translation
    raise FileNotFoundError(ENOENT,
                          'No translation file found for domain', domain)
FileNotFoundError: [Errno 2] No translation file found for domain: 'messages'
```

```
t = gettext.translation(
    'messages', localedir='locale', fallback=True
)
```

Retorna o texto original se não
encontrar a língua



Caso você tenha que debugar algo no compilado, você pode usar **msgunfmt**, que vai mostrar o texto original. É bom em momentos de debug

```
msgunfmt \
  locale/pt_BR/ LC_MESSAGES/messages.mo
```

Exemplos de projetos com .PO



- Kdenlive - Editor de vídeo em C++
- Humanize - Biblioteca em Python
- Deluge - Cliente de torrent em Python
- Openshot - Editor de vídeo em Python

Novas mensagens e agora?



Como toda aplicação está em constante desenvolvimento, é importante saber o que fazer quando novas mensagens surgirem.

Quando adicionamos novos gettexts, precisamos do **msgmerge**:

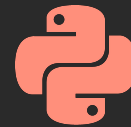
```
msgmerge -U pt_BR.po messages.pot
```

Isso vai adicionar as novas mensagens ao portable que já temos. Isso evita perder as traduções que já foram feitas.

Aplicando gettext na nossa aplicação



Parte prática



Trabalhando com plurais



Um dos pontos mais importantes quando trabalhando com texto é entender que mensagens nem sempre são "fixas". Elas se modificam em relação ao contexto em que a mensagem é exibida. O caso mais clássico dessas modificações são os plurais.

Vamos imaginar uma caixa de mensagens:

O texto muda se você não tem mensagens, se tem uma mensagem ou se tem N mensagens.

No momento de escrever um código internacionalizado, essas regras mudam. Principalmente pelo fato de plurais não serem iguais em todas as línguas.

Trabalhando com plurais



```
number_of_messages = 1

if not number_of_messages:
    print(_('No new messages'))
else:
    print(
        t.ngettext(
            'You have %(count)d message',
            'You have %(count)d messages',
            number_of_messages
        ) % {'count': number_of_messages}
    )
```

Trabalhando com plurais



```
number_of_messages = 1

if not number_of_messages:
    print(_('No new messages'))
else:
    print(
        t.ngettext(
            'You have %(count)d message',
            'You have %(count)d messages',
            number_of_messages
        ) % {'count': number_of_messages}
    )
```

Singular

Valor de
variação

Plural



Formatadores para mensagens

Toda vez que precisarmos exibir um valor na string, temos os formatadores:

- **%s: strings**
- **%d: números**
- **%f: floats**
- **%x: hexa**

Evite usar diretamente a variável como:

- "Você tem **%d** mensagens"

Isso dificulta a tradução, quem traduzir tem que entender o contexto. **Use**

- "Você tem **%(contagem)d** mensagens"

Algo que dê contexto a quem for traduzir

Arquivo .pot



O que vai nos gerar um `.pot` como esse:

```
Plural-Forms: nplurals=INTEGER; plural=EXPRESSION
```

```
#: exemplo_00.py:10
```

```
msgid "No new messages"
```

```
msgstr ""
```

```
#: exemplo_00.py:14
```

```
#, python-format
```

```
msgid "You have %(count)d message"
```

```
msgid_plural "You have %(count)d message"
```

```
msgstr[0] ""
```

```
msgstr[1] ""
```

msgid é usado no singular, **msgid_plural** para o plural

As variações nas variações...



Nesse momento, o ponto de atenção é o "Plural-Forms":

Plurais são diferentes em línguas diferentes

Vamos para o exemplo do Russo:

- **Singular**[0]: todo número terminado em 1, mas não em 11, usa a forma singular: 1, 21, 31, 41, ...
- **Paucal**[1]: usada quando o número termina em 2, 3 ou 4, mas não em 12 a 14: 2, 3, 4, 22, 23, 24 ...
- **Geral**[2]: usada para todos os outros casos. Exemplos: 0, 5-20, 11-14, 25, 100, 111, etc.

```
n%10==1 && n%100!=11 ? 0 : n%10>=2 && n%10<=4 && (n%100<10 || n%100>=20) ? 1 : 2
```



O que pode gerar uns .po diferentes

Onde cada número da expressão cai em um **msgstr** diferente (ps não sei russo):

```
msgid "You have %(count)d new message"
msgid_plural "You have %(count)d new messages"
msgstr[0] "У вас %(count)d новое сообщение"
msgstr[1] "У вас %(count)d новых сообщения"
msgstr[2] "У вас %(count)d новых сообщений"
```

Aplicando **ngettext** na nossa aplicação



Parte prática



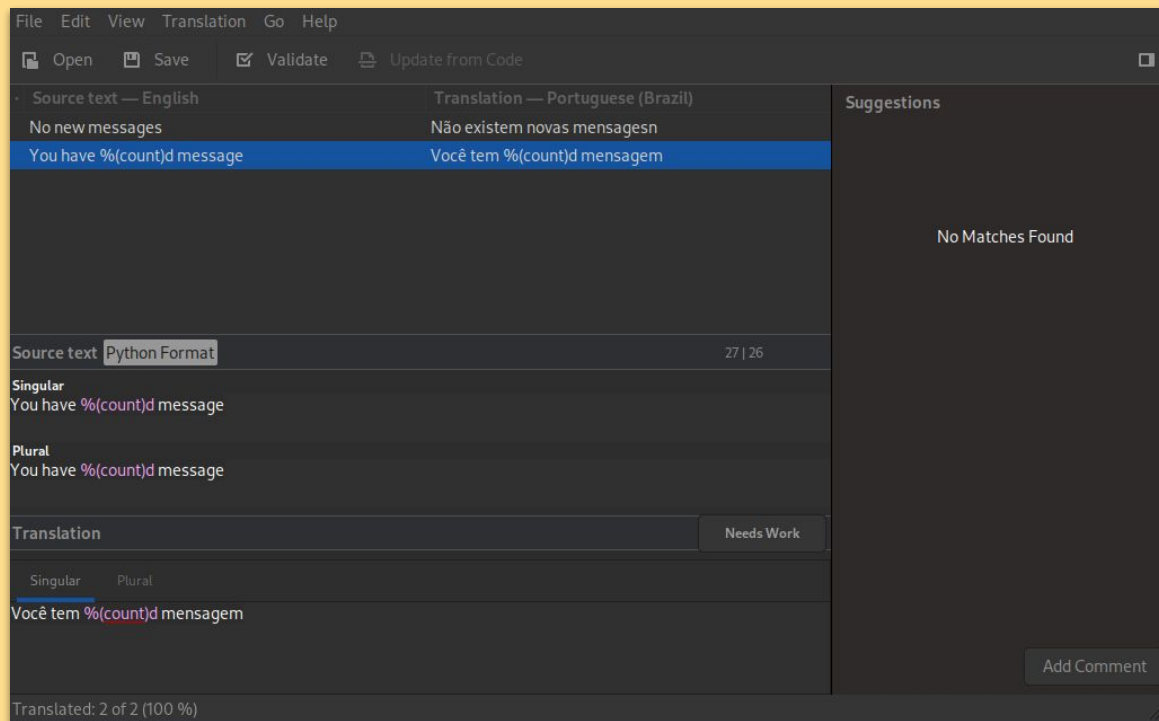
Extras

Algumas coisas
mais

Trabalhando com arquivos .po em GUI



Existem algumas aplicações que suportam .po e .pot para tradução em interface gráfica, o que evita um certo transtorno para pessoas não técnicas. Entre elas destaco o **poedit**



Continuous Translate



Uma das coisas importantes sobre trabalhar com um software multilíngue é contar com a colaboração de pessoas, técnicas ou não, para traduzir para as línguas que não conhecemos.

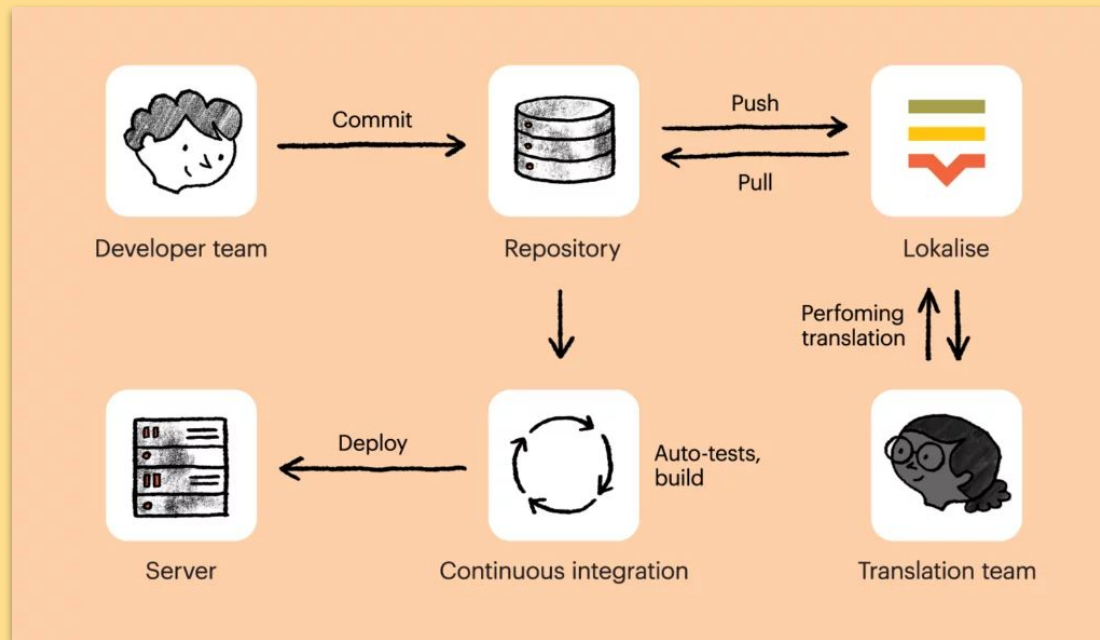
Como o .po é um padrão amplamente aceito, diversas ferramentas de tradução colaborativa aceitam ele:

- Transifex: Usada para tradução da documentação do **python**
- weblate: Usada na tradução do **capypreader**
- gitlocalize: A que vamos usar por ser ridiculamente simples :)
- ... e muito mais

Continuous Translate



A ideia da tradução contínua é poder ir traduzindo em relação as atualizações do repositório. Sempre integrando novas traduções para coisas novas



<https://lokalise.com/blog/continuous-localization-101/>

Aplicando **CT** na nossa aplicação



Parte prática



Onde colocar as traduções no pacote?



Não existe uma regra clara para isso. Me baseando no **humanize**. Acredito que uma arquitetura viável seja a raiz do pacote, não do repositório. Assim ficam acessíveis globalmente (posso estar errado):

```
src
├── pacote
│   ├── app.py
│   └── locale
│       ├── pt_BR
│       │   └── LC_MESSAGES
│       │       └── messages.po
```

Builds de pacotes



Um problema comum é disponibilizar as traduções (.mo) no pacote, por não serem .py as ferramentas de build costumam ignorar esses arquivos.

```
# Exemplo setuptools
[tool.setuptools.packages.find]
where = ["pacote"]

[tool.setuptools.package-data]
"*" = ["*.mo"]
```

Compilar .po -> .mo no pacote?



Existe uma discussão interessante sobre isso. Sobre como fazer o setuptools fazer um build dos .mo durante a compilação do pacote.

Não vou me estender aqui... Mas é uma coisa que você deveria olhar :)

<https://github.com/pypa/setuptools/discussions/3912>

outra ideia é criar um script, o humanize tem um um exemplo:

<https://github.com/python-humanize/humanize/tree/main/scripts>

Referências

- **ESSELINK, Bert; ESSELINK, Bert (ORGS.). A Practical guide to localization.** Amsterdam Philadelphia: John Benjamins Pub. Co, 2000.
- **FREE SOFTWARE FOUNDATION. Top (GNU gettext utilities).** , [S.d.]. Disponível em: <<https://www.gnu.org/software/gettext/manual/gettext.html>>. Acesso em: 15 jul. 2025
- **FREE STANDARDS GROUP. OpenI18N 1.3.** , 2003. Disponível em: <<http://openi18n.web.fc2.com/numa/OpenI18N1.3.pdf>>
- **I18NGUY. Origin Of The Abbreviation I18n For Internationalization.** Disponível em: <<http://www.i18nguy.com/origini18n.html>>. Acesso em: 16 jul. 2025.
- **Language Plural Rules.** Disponível em: <https://www.unicode.org/cldr/charts/47/supplemental/language_plural_rules.html>. Acesso em: 21 jul. 2025.
- **LINUX STANDARD BASE. The Open Group Base Specifications Issue 8,** 2018 edition. Disponível em: <<https://pubs.opengroup.org/onlinepubs/9799919799/nframe.html>>. Acesso em: 20 jul. 2025.
- **PYTHON SOFTWARE FOUNDATION. gettext** — Serviços de internacionalização multilíngues. Disponível em: <<https://docs.python.org/pt-br/3.13/library/gettext.html>>. Acesso em: 15 jul. 2025b.
- **PYTHON SOFTWARE FOUNDATION. Internacionalização.** Documentação. Disponível em: <<https://docs.python.org/3/library/i18n.html>>. Acesso em: 20 jul. 2025c.
- **VEVERIS, Emils. Continuous localization & translation:** what is it & how to do it. Lokalise Blog, 4 nov. 2022. Disponível em: <<https://lokalise.com/blog/continuous-localization-101/>>. Acesso em: 26 jul. 2025
- **W3C. About W3C Internationalization (i18n).** Disponível em: <<https://www.w3.org/International/i18n-drafts/nav/about>>. Acesso em: 16 jul. 2025a.
- **W3C. Localization vs. Internationalization.** Disponível em: <<https://www.w3.org/International/questions/qa-i18n>>. Acesso em: 16 jul. 2025b.



apoia.se/livedepython



pix.dunossauro@gmail.com



patreon.com/dunossauro



Ajude o projeto <3

