Recunchos de Python

M. Torre Castro[1]

[1]Python Coruña

Charlas Python Coruña, 2024

Índice

Outline

Separadores visuales

```
1  >>> a = 1000000000000000 # cantos 0s hai
2
3  >>> b = 1_000_000_000_000_000 # así mellor, non sí?
4
5  >>> print(a == b)
6  True
7
8  >>> 1e10
9  10000000000.0
```

Problemas con redondeos

```
1  >>> 1.2 + 2.2
2  3.4000000000000004
3  >>>
4  >>> 0.1 + 0.1 + 0.1 - 0.3
5  5.551115123125783e-17
```

Decimal ao rescate

```
1  from decimal import Decimal
2  >>> Decimal(1.2) + Decimal(2.2)
3  Decimal('3.4')
4
5  >>> d1 = Decimal(0.1)
6  >>> d = d1 + d1 + d1 - Decimal(0.3)
7  Decimal('2.77555756156515654042363166 8E-17')
8
9  >>> round(float(d))
10 0
```

Fraction ao rescate

```
1  >>> 0.2 + 0.1
2  0.30000000000000004
3  >>>
4  >>> from fractions import Fraction
5  >>> Fraction(2, 10) + Fraction(1, 10)
6  Fraction(3, 10)
7  >>> float(Fraction(3, 10))
8  0.3
```

Prefixos e sufixos

```
1  # Python >= 3.9
2  # Ata agora o mellor era usar slicing, menos lexible
3
4  >>> "three␣features␣in␣Python".removesuffix("␣Python")
5  'three␣features␣in'
6
7  >>> "three␣features␣in␣Python".removeprefix("three␣")
8  'features␣in␣Python'
9
10 >>> "three␣features␣in␣Python".removeprefix("")
11 'three␣features␣in␣Python'
```

F-strings con formateo I

```
1  # Python >= 3.6
2  # f'{value:{align}{width}}'
3
4  >>> import math
5
6  >>> answer = 42
7  >>> print(f'{answer:4d}')
8  '  42'
9
10 >>> print(f'{answer:04d}')
11 '0042'
12
13 >>> print(f'{math.pi:06.2f}')
14 '003.14'
```

F-strings con formateo II

```
>>> import math
>>> print(f'{math.pi}')
3.141592653589793

>>> print(f'{math.pi:_<25}')
3.141592653589793________

>>> print(f'{math.pi:_>25}')
________3.141592653589793

>>> print(f'{math.pi:_^25}')
____3.141592653589793____

>>> print(f'{math.pi:_^25.5}')
_________3.1416__________
```

F-strings con formateo III

```
1  >>> from datetime import datetime
2  >>> dt = datetime(2022, 4, 11, 13, 37)
3  '2022-04-11 13:37'
4
5  >>> dfmt = "%Y-%m-%d"
6  >>> tfmt = "%H:%M"
7
8  >>> print('{dt:{dfmt} {tfmt}}')
9  '2022-04-11 13:37'
```

Outline

Variable de descarte

- En ocasións, temos expresións que devolven varios valores, pero non usamos todos
- Nestes casos, está recomendado non gardar os valores que non usamos
- En Python temos a expresión _ ou ___ pra recoller valores que non usamos e non ter que crear unha variable

Variable de descarte

- En ocasións, temos expresións que devolven varios valores, pero non usamos todos
- Nestes casos, está recomendado non gardar os valores que non usamos
- En Python temos a expresión _ ou ___ pra recoller valores que non usamos e non ter que crear unha variable

Variable de descarte

- En ocasións, temos expresións que devolven varios valores, pero non usamos todos
- Nestes casos, está recomendado non gardar os valores que non usamos
- En Python temos a expresión _ ou ___ pra recoller valores que non usamos e non ter que crear unha variable

Variables de descarte

```
1  for _ in (1, 2, 3):
2      print('*' * 10)  # 3 veces
3
4  x, _, z = (1, 2, 3)
5  x, z
6
7  # 1, 3
8
9  def callback_handler(__):
10     print('Manexando␣evento')
```

Outline

Ellipsis

Ellipsis

- Ellipsis é unha expresión especial en Python representada por: ...
- Pode usarse do mesmo xeito que pass pra denotar que unha función non fai nada
- Tamén pode usarse en type hinting

Ellipsis

Ellipsis

- Ellipsis é unha expresión especial en Python representada por: ...
- Pode usarse do mesmo xeito que pass pra denotar que unha función non fai nada
- Tamén pode usarse en type hinting

Ellipsis

Ellipsis

- Ellipsis é unha expresión especial en Python representada por: ...
- Pode usarse do mesmo xeito que pass pra denotar que unha función non fai nada
- Tamén pode usarse en type hinting

Ellipsis I

```
1  def do_nothing():
2      ...
3
4  numbers: tuple[int, ...]
5  # Allowed:
6  numbers = ()
7  numbers = (1,)
8  numbers = (4, 5, 6, 99)
9
10 # Not allowed:
11 numbers = (1, "a")
12 numbers = [1, 3]
```

Ellipsis II

```
1  def calculate(i: int,
2                action: Callable[..., int],
3                *args: int) -> int:
4      return action(i, *args)
5
6  # Works:
7  calculate(1, add_one)
8  calculate(1, multiply_with, 3)
9
10 # Doesn't work:
11 calculate(1, 3)
12 calculate(1, as_pixels)
```

Outline

Orde nos diccionarios

```
>>> {"one": 1, "two": 2, "three": 3}  # CPython <= 3.5
{'three': 3, 'one': 1, 'two': 2}

>>> {"one": 1, "two": 2, "three": 3}  # CPython >= 3.6
{'one': 1, 'two': 2, 'three': 3}

# Guaranteed in python 3.7
```

Unión de diccionarios

```
>>> pycon = {2016: "Portland", 2018: "Cleveland"}
>>> europython = { 2017: "Rimini",
                   2018: "Edinburgh",
                   2019: "Basel" }

>>> pycon | europython     # devolve a union de diccionarios
{ 2016: 'Portland',
  2018: 'Edinburgh',
  2017: 'Rimini',
  2019: 'Basel' }

>>> pycon |= europython    # actualiza o diccionario pycon
>>> pycon
{ 2016: 'Portland',
  2018: 'Edinburgh',
  2017: 'Rimini',
  2019: 'Basel' }
```

Estructuras por comprensión

```
1  # Cada liña ten prezo e nome separado por coma ex:
2  # '12,Pizza margherita'
3  with open('menu.txt') as menu_file:
4      lines = menu_file.readlines()
5
6  # Lista de pratos ofertados
7  dishes = [line.split(',')[1] for line in lines]
8
9  # Diccionario con clave prato e valor prezo
10 dish_dict { line.split(',')[1]: float(line.split(',')[0])
11             for line in lines }
12
13 # Conxunto de todos os prezos
14 # (ex: analizar prezo pedido mínimo
15 # ou ofrecer pratos máis económicos)
16 prices = { float(line.split(',')[1]) for line in lines }
```

Estructuras por comprensión: Filtrados

```
1  dishes_with_long_names = [ dish
2                             for dish in dishes
3                             if len(dish) > 10 ]
4
5  too_expensive_dishes = [dish
6                          for dish,price in dish_dict
7                          if price > 25]
8
9  too_expensive_prices = { price
10                          for price in prices
11                          if price > 25 }
```

Xeneradores e cuantificadores condicionais

Cando tes moitas condicións que comprobar

- Tralas estructuras definidas por comprensión, é fácil entender os xeneradores por comprensión
- Podemos crealos sustituendo os delimitadores por parénteses
- Son utilizables como iterables (ex: con for) e podense usar con funcións que usen iterables
- Son máis eficientes en memoria

Xeneradores e cuantificadores condicionais

Cando tes moitas condicións que comprobar

- Tralas estructuras definidas por comprensión, é fácil entender os xeneradores por comprensión
- Podemos crealos sustituendo os delimitadores por parénteses
- Son utilizables como iterables (ex: con for) e podense usar con funcións que usen iterables
- Son máis eficientes en memoria

Xeneradores e cuantificadores condicionais

Cando tes moitas condicións que comprobar

- Tralas estructuras definidas por comprensión, é fácil entender os xeneradores por comprensión
- Podemos crealos sustituendo os delimitadores por parénteses
- Son utilizables como iterables (ex: con for) e podense usar con funcións que usen iterables
- Son máis eficientes en memoria

Xeneradores e cuantificadores condicionais

Cando tes moitas condicións que comprobar

- Tralas estructuras definidas por comprensión, é fácil entender os xeneradores por comprensión
- Podemos crealos sustituendo os delimitadores por parénteses
- Son utilizables como iterables (ex: con for) e podense usar con funcións que usen iterables
- Son máis eficientes en memoria

Xeneradores

```
1  # Iterar sobre numeros pares de 0 a 10000
2  for x in (num_par for num_par in range(0,10_000,2)):
3      # Facer cousas con x
4
5
6  # Sumar os pares de 0 a 10000
7  sum(num_par for num_par in range(0,10_000,2))
8
9
10 # Deste xeito non queda a lista ocupando espazo en memoria
```

Cuantificadores condicionais: All

Un para todos e todos para un!

```
1  kids_by_age = {
2      'Ana': 12,
3      'Pablo': 8,
4      'Veronica': 15,
5      'David': 16,
6      'Noa': 14,
7      # ... poderían ser decenas ...
8  }
9
10 # Se hai que escoller unha película pra todos,
11 # ¿podemos coller unha calificada pra maiores de 13?
12
13 over_13_is_ok = all(age > 13 for kid,age in kids_by_age.items())
14 over_7_is_ok = all(age > 7 for kid,age in kids_by_age.items())
```

Cuantificadores condicionais: Any

Hai alguén aí?

```
1   candidates = [
2       { "name": "Brais", "xp": 1, "degree": True "langs": ["c++", "ada"] },
3       { "name": "Breixo","xp": 3, "degree": True, "langs": ["python", "java"] },
4       { "name": "Sara", "xp": 5, "degree": False, "langs": ["js", "ada"] },
5       { "name": "Antía", "xp": 2, "degree": False, "langs": ["python", "ruby"] },
6       # ...poderian ser decenas...
7   ]
8
9   def with_degree(candidate: dict):
10      return candidate['degree']
11
12  def knows_python(candidate: dict):
13      return 'python' in candidate['langs'] and candidate['xp'] > 2
14
15  # Comprobamos 2 condicions, pero poderían ser 10
16  # Isto é ineficiente, pero serve como exemplo
17  if any(knows_python(c) or with_degree(c) for c in candidates):
18      tuple(send_email(person)
19            for person in candidates
20            if any( (knows_python(person), with_degree(person))
21      )
```

Vida máis aló de list e dict

Coleccións interesantes de collections

Deque "Lista" eficiente pra moitas insercióǹs e borrados, ambas polos extremos

Counter Diccionario útil pra realizar conteos de ocurrencias (semellante a defaultdic(int))

defaultdict Diccionario con todos ou algúns valores por defecto (dependendo da clave)

namedtuple Función que permite crear unha tupla con campos nomeados

Vida máis aló de list e dict

Coleccións interesantes de collections

Deque "Lista" eficiente pra moitas insercióǹs e borrados, ambas polos extremos

Counter Diccionario útil pra realizar conteos de ocurrencias (semellante a defaultdic(int))

defaultdict Diccionario con todos ou algúns valores por defecto (dependendo da clave)

namedtuple Función que permite crear unha tupla con campos nomeados

Vida máis aló de list e dict

Coleccións interesantes de collections

Deque "Lista" eficiente pra moitas insercióǹs e borrados, ambas polos extremos

Counter Diccionario útil pra realizar conteos de ocurrencias (semellante a defaultdic(int))

defaultdict Diccionario con todos ou algúns valores por defecto (dependendo da clave)

namedtuple Función que permite crear unha tupla con campos nomeados

Vida máis aló de list e dict

Coleccións interesantes de collections

Deque "Lista" eficiente pra moitas insercióǹs e borrados, ambas polos extremos

Counter Diccionario útil pra realizar conteos de ocurrencias (semellante a defaultdic(int))

defaultdict Diccionario con todos ou algúns valores por defecto (dependendo da clave)

namedtuple Función que permite crear unha tupla con campos nomeados

Outline

O operador morsa

Walrus operator

- Disponible desde Python 3.8
- É un operador de asignación de **nova variable** utilizable dentro dun contexto de non asignación
- Devolve un valor pero non pode ser usado como único operador
- Axuda a evitar a repetición en:
  - Chamadas de funcións
  - Liñas / Instruccións
  - Iteradores

O operador morsa

Walrus operator

- Disponible desde Python 3.8
- É un operador de asignación de **nova variable** utilizable dentro dun contexto de non asignación
- Devolve un valor pero non pode ser usado como único operador
- Axuda a evitar a repetición en:
  - Chamadas de funcións
  - Liñas / Instruccións
  - Iteradores

O operador morsa

Walrus operator

- Disponible desde Python 3.8
- É un operador de asignación de **nova variable** utilizable dentro dun contexto de non asignación
- Devolve un valor pero non pode ser usado como único operador
- Axuda a evitar a repetición en:
  - Chamadas de funcións
  - Liñas / Instruccións
  - Iteradores

O operador morsa

Walrus operator

- Disponible desde Python 3.8
- É un operador de asignación de **nova variable** utilizable dentro dun contexto de non asignación
- Devolve un valor pero non pode ser usado como único operador
- Axuda a evitar a repetición en:
  - Chamadas de funcións
  - Liñas / Instruccións
  - Iteradores

O operador morsa

Walrus operator

- Disponible desde Python 3.8
- É un operador de asignación de **nova variable** utilizable dentro dun contexto de non asignación
- Devolve un valor pero non pode ser usado como único operador
- Axuda a evitar a repetición en:
  - Chamadas de funcións
  - Liñas / Instruccións
  - Iteradores

O operador morsa

Walrus operator

- Disponible desde Python 3.8
- É un operador de asignación de **nova variable** utilizable dentro dun contexto de non asignación
- Devolve un valor pero non pode ser usado como único operador
- Axuda a evitar a repetición en:
  - Chamadas de funcións
  - Liñas / Instruccións
  - Iteradores

O operador morsa

Walrus operator

- Disponible desde Python 3.8
- É un operador de asignación de **nova variable** utilizable dentro dun contexto de non asignación
- Devolve un valor pero non pode ser usado como único operador
- Axuda a evitar a repetición en:
  - Chamadas de funcións
  - Liñas / Instruccións
  - Iteradores

Operador morsa: Exemplo código

```
1  >>> value = False
2  >>> value
3  False
4
5  >>> value := True
6    File "<stdin>", line 1
7      value := True
8            ^
9  SyntaxError: invalid syntax
10
11 >>> (value := True) # Válido, pero non recomendado
12 True
```

Operador morsa: Exemplo código II

```
1  # Asignacion no if
2  number = 3
3  if square := number ** 2 > 5:
4      print(square)
5
6  # Petición de entrada de usuario
7  while (command := input("> ")) != "quit":
8      print("You entered:", command)
```

Operador morsa: Exemplo código III

```
1  condition_list = [line.strip() == '' for line in lines]
2  if all(condition_list):
3      print("All lines are blank")
4  else:
5      print("Not all lines are blank")
6
7
8  if all((nonblank := line).strip() == '' for line in lines):
9      print("All lines are blank")
10 else:
11     print("First non-blank line:", nonblank)
```

Operador morsa: Exemplo código IV

```
1  while True:
2      old = total
3      total += term
4      if old == total:
5          return total
6      term *= mx2 / (i*(i+1))
7      i += 2
8
9  # versus
10
11 while total != (total := total + term):
12     term *= mx2 / (i*(i+1))
13     i += 2
14     return total
```

Operador morsa: Exemplo código V

```
1  reductor = dispatch_table.get(cls)
2  if reductor:
3      rv = reductor(x)
4  else:
5      reductor = getattr(x, "__reduce_ex__", None)
6
7      if reductor:
8          rv = reductor(4)
9      else:
10         reductor = getattr(x, "__reduce__", None)
11
12         if reductor:
13             rv = reductor()
14         else:
15             raise Error("Mensaxe de erro")
```

Operador morsa: Exemplo código VI

```
1  if reductor := dispatch_table.get(cls):
2      rv = reductor(x)
3  elif reductor := getattr(x, "__reduce_ex__", None):
4      rv = reductor(4)
5  elif reductor := getattr(x, "__reduce__", None):
6      rv = reductor()
7  else:
8      raise Error("Mensaxe de erro")
```

Non abusar do operador morsa

Evitar adiccióńs

- Tralo visto, vemos que o operador morsa ten o seu ámbito de utilidade
- Pero hai que ter coidado de non querer usalo en todas partes e facer liñas moi longas
- Lembrade que a lexibilidade e a elegancia son representativos de Python

Non abusar do operador morsa

Evitar adiccióńs

- Tralo visto, vemos que o operador morsa ten o seu ámbito de utilidade
- Pero hai que ter coidado de non querer usalo en todas partes e facer liñas moi longas
- Lembrade que a lexibilidade e a elegancia son representativos de Python

Non abusar do operador morsa

Evitar adiccióńs

- Tralo visto, vemos que o operador morsa ten o seu ámbito de utilidade
- Pero hai que ter coidado de non querer usalo en todas partes e facer liñas moi longas
- Lembrade que a lexibilidade e a elegancia son representativos de Python

Outline

*Args, **kwargs

- Esta sintaxe permítenos pasar multiples argumentos por posición ou por pares clave=valor
- Definir ambos valores é unha boa práctica en librerías e funcións que ofrecemos a outros programadores
- A sintaxe de exemplo sería con eles sempre ao final.

Ex: def func(x, y, *args, **kwargs)

*Args, **kwargs

- Esta sintaxe permítenos pasar multiples argumentos por posición ou por pares clave=valor
- Definir ambos valores é unha boa práctica en librerías e funcións que ofrecemos a outros programadores
- A sintaxe de exemplo sería con eles sempre ao final.

Ex: def func(x, y, *args, **kwargs)

*Args, **kwargs

- Esta sintaxe permítenos pasar multiples argumentos por posición ou por pares clave=valor
- Definir ambos valores é unha boa práctica en librerías e funcións que ofrecemos a outros programadores
- A sintaxe de exemplo sería con eles sempre ao final.

Ex: `def func(x, y, *args, **kwargs)`

*Args, **kwargs

- Esta sintaxe permítenos pasar multiples argumentos por posición ou por pares clave=valor
- Definir ambos valores é unha boa práctica en librerías e funcións que ofrecemos a outros programadores
- A sintaxe de exemplo sería con eles sempre ao final.

Ex: def func(x, y, *args, **kwargs)

*args

```
1  def my_sum(my_integers: list):
2      result = 0
3      for x in my_integers:
4          result += x
5      return result
6
7  print(my_sum([1, 2, 3]))
8
9
10 def my_sum(*args):
11     result = 0
12     # Iterating over the Python args tuple
13     for x in args:
14         result += x
15     return result
16
17 print(my_sum(1, 2, 3))
```

**kwargs

```
1  def concatenate(**kwargs):
2      result = ""
3      # Iterating over the Python kwargs dictionary
4      for arg in kwargs.values():
5          result += arg
6      return result
7
8  print(concatenate(a="Real", b="Python", c="Is", d="Great", e="!"))
9
10 def area(base, height):
11     return b * h / 2
12
13 triangle = {
14     'base': 10,
15     'height' 20
16 }
17
18 print(area(**triangle))
```

Firmas de métodos

- A maioría de nós coñecemos como é a firma ou cabecera dun método, que indica o nome e parámetros/argumentos
- Como sabemos, ao chamar unha función podemos especificar o argumento por posición ou por nome
- Pero en Python **3.8** introdúxose unha característica nova pra permitir forzar o tipo de parámetro desexado

Firmas de métodos

- A maioría de nós coñecemos como é a firma ou cabecera dun método, que indica o nome e parámetros/argumentos
- Como sabemos, ao chamar unha función podemos especificar o argumento por posición ou por nome
- Pero en Python **3.8** introdúxose unha característica nova pra permitir forzar o tipo de parámetro desexado

Firmas de métodos

- A maioría de nós coñecemos como é a firma ou cabecera dun método, que indica o nome e parámetros/argumentos
- Como sabemos, ao chamar unha función podemos especificar o argumento por posición ou por nome
- Pero en Python **3.8** introdúxose unha característica nova pra permitir forzar o tipo de parámetro desexado

Argumentos por posición

```
1  def tofloat(x):
2      return float(x)
3
4  y = tofloat(3.8)
5  print(y)
6
7  y = tofloat(x=3.8)
8  print(y)
```

Argumentos só por posición

```
1  def tofloat(x, /): # Python 3.8
2      return float(x)
3
4  y = tofloat(3.8)
5  print(y)
6
7  y = tofloat(x=3.8)
8  # TypeError: tofloat() got some positional-only arguments
9                passed as keyword arguments: 'x'
```

Argumentos só por nome

```
>>> def to_fahrenheit(*, celsius):
...     return 32 + celsius * 9 / 5
...
>>> to_fahrenheit(40)

Traceback (most recent call last):
File "<stdin>", line 1, in <module>
TypeError: to_fahrenheit() takes 0 positional arguments
           but 1 was given

>>> to_fahrenheit(celsius=40)
104.0
```

Tipos de Parámetros

Esquema

```
1  def f(pos1, pos2, /, pos_or_kwd, *, kwd1, kwd2):
2        -----------    ----------     ----------
3        |              |                |
4        |       Positional or keyword   |
5        |                                - Keyword only
6         -- Positional only
```

Combinación

Nome e posición obrigados

```
1  >>> def f(text, /, border="-", *args, width=50, **kargs):
2  ...     return f" {text} ".center(width, border)
3  ...
4
5  >>> f("Positional-only Arguments")
6  '----------- Positional-only Arguments ------------'
7
8  >>> headline(text="This doesn't work!")
9
10 Traceback (most recent call last):
11 File "<stdin>", line 1, in <module>
12 TypeError: headline() got some positional-only arguments
13            passed as keyword arguments: 'text'
```

Outline

Switch / case en Python?

Non deixa indiferente

- Python >= 3.10
- Fai encaixar literales con patróns
- Detecta e deconstrue estructuras de datos
- Usa diferentes tipos de patróns

Switch / case en Python?

Non deixa indiferente

- Python >= 3.10
- Fai encaixar literales con patróns
- Detecta e deconstrue estructuras de datos
- Usa diferentes tipos de patróns

Switch / case en Python?

Non deixa indiferente

- Python >= 3.10
- Fai encaixar literales con patróns
- Detecta e deconstrue estructuras de datos
- Usa diferentes tipos de patróns

Switch / case en Python?

Non deixa indiferente

- Python >= 3.10
- Fai encaixar literales con patróns
- Detecta e deconstrue estructuras de datos
- Usa diferentes tipos de patróns

Literales

```
1  def greet(name):
2      match name:
3          case "Guido":
4              print("Ola, Guido! Grazas por crear Python")
5          case "Python Coruña":
6              print("Benvidos todos a esta nova charla")
7          case _:
8              print("Boas, que podo facer por vostede?")
```

Deconstruindo estructuras

```
>>> user = {
...     "name": {
...         "first": "Pablo",
...         "last": "Galindo Salgado"
...     },
...     "title": "Python 3.10 release manager",
... }

>>> match user:
...     case {"name": {"first": first_name}}:
...         pass
...
>>> first_name
'Pablo'
```

Usando patróns I

```
1  def sum_list(numbers: list):
2      match numbers:
3          case []:
4              return 0
5          case [first, *rest]:
6              return first + sum_list(rest)
```

Usando patróns II

```
1  def sum_list(numbers: list):
2      match numbers:
3          case []:
4              return 0
5          case [int(first) | float(first), *rest]:
6              return first + sum_list(rest)
7          case _:
8              raise ValueError(f"So pode sumar numeros")
```

Fizzbuzz I

```
1  def fizzbuzz(number):
2      is_mod_3 = number % 3 == 0
3      is_mod_5 = number % 5 == 0
4
5      if is_mod_3 and is_mod_5:
6          return "fizzbuzz"
7      elif is_mod_3:
8          return "fizz"
9      elif is_mod_5:
10         return "buzz"
11     else:
12         return str(number)
```

Fizzbuzz II

```
1  def fizzbuzz(number):
2      is_mod_3 = number % 3 == 0
3      is_mod_5 = number % 5 == 0
4
5      match (is_mod_3, is_mod_5):
6          case (True, True):
7              return "fizzbuzz"
8          case (True, _):
9              return "fizz"
10         case (_, True):
11             return "buzz"
12         case _:
13             return str(number)
```

Outline

Cousas de interese

zipapp Librería pra construir un paquete executable "standalone" con dependencias cun interprete python

webbrowser Modulo pra abrir o navegador desde Python

importlib.resources Modulo pra importar ficheiros de datos

Cousas de interese

zipapp Librería pra construir un paquete executable "standalone" con dependencias cun interprete python

webbrowser Modulo pra abrir o navegador desde Python

importlib.resources Modulo pra importar ficheiros de datos

Cousas de interese

zipapp Librería pra construir un paquete executable "standalone" con dependencias cun interprete python

webbrowser Modulo pra abrir o navegador desde Python

importlib.resources Modulo pra importar ficheiros de datos

webbrowser

```
1  >>> import webbrowser
2  >>> webbrowser.open_new('https://trickster.dev')
3  True
4
5  >>> import os
6  >>> file_url = 'file://' + os.path.realpath('test.html')
7  >>> webbrowser.open(file_url)
8  True
```

importlib.resources I

Solucións ata agora

- Hard-code. Poñer de forma estática os valores das rutas. Non portable
- Referenciar as rutas dos ficheiros de datos desde paquetes usando ___file___. Non funciona dentro dun zip
- Usar setuptools.pkg_resources pra acceder ao ficheiro de datos. Moi lento

importlib.resources I

Solucións ata agora

- Hard-code. Poñer de forma estática os valores das rutas. Non portable
- Referenciar as rutas dos ficheiros de datos desde paquetes usando ___file___. Non funciona dentro dun zip
- Usar setuptools.pkg_resources pra acceder ao ficheiro de datos. Moi lento

importlib.resources I

Solucións ata agora

- Hard-code. Poñer de forma estática os valores das rutas. Non portable
- Referenciar as rutas dos ficheiros de datos desde paquetes usando ___file___. Non funciona dentro dun zip
- Usar setuptools.pkg_resources pra acceder ao ficheiro de datos. Moi lento

importlib.resources II

```
"""
data/
     alice.txt
     __init__.py
"""

>>> from importlib import resources  # python >= 3.7
>>> with resources.open_text("data", "alice.txt") as fid:
...     alice = fid.readlines()
...
>>> print("".join(alice[:7]))
CHAPTER I. Down the Rabbit-Hole
Alice was beginning to get very tired of sitting by her
sister on the bank, and of having nothing to do: once or
...
```

Resumo

- Sempre hai detalliños interesantes que non coñecemos nas linguaxes
- Algúns deles poden adecuarse aos nosos casos de uso

Resumo

- Sempre hai detalliños interesantes que non coñecemos nas linguaxes
- Algúns deles poden adecuarse aos nosos casos de uso

Pra ler máis I

- https://docs.python.org/3/library/string.html#format-string-syntax
- https://fstring.help/cheat/
- https://docs.python.org/3/library/collections.html
- https://docs.python.org/3/library/zipapp.html
- https://docs.python.org/3/library/webbrowser.html
- https://realpython.com/python-kwargs-and-args/
- https://realpython.com/python-ellipsis/