

Python - P24

F09 - enumerate

enumerate (iterable, start=0) 将传入的可迭代对象组成成(下标, 元素)的元组形式

主要用于在for循环中遍历元素及元素对应的下标

如 for ind, element in enumerate('abcd'): →
print(ind, element)

iterable 参数传入一个可迭代对象

要求是序列(sequence), 迭代器(iterator)或支持迭代(iteration)的对象

start 参数传入一个整型值, 用作下标的起始值.

如 for ind, element in enumerate('abcd', start=-2): →
print(ind, element)

start 参数的默认值为0, 即下标编号从0开始

下标的生成遵循 auto increment 的方式, 即自动加1

enumerate() 返回一个形如 (index, element) 的迭代器, 支持 next() 方法

myEnumerate [] = [] → myEnumerate :: [a] → [(Int, a)]

myEnumerate lst = [] → myEnumerate :: [a] → [(Int, a)]

let func n [] acc = reverse acc

def func n (x:xs) acc = func (n+1) xs (c(n,x):acc)

in func 0 lst []

filter (function, iterable), 利用传入的函数筛选传入的可迭代对象

主要用于根据给定的判断函数筛选给定的可迭代对象中的元素

如 list(filter(lambda x : x%2==0, [1, 2, 3, 4, 5])) → [2, 4]

对于 function, 要求传入一个形如 a → Bool 的函数

其返回值为 TRUE / FALSE 用于决定是否保留当前元素

判断条件相对简单而可以用一个表达式计算得出

常用 lambda 函数的形式传入参数

注意 Python 的 filter() 函数允许向 function 参数传入 None

当 function = None 时, filter 函数会使用 bool() 作为替代

即对每个元素进行真值检查(truth value testing)以决定是否保留

如 list(filter(None, [None, 1, 0, 1, 2, 3])) → [1, 2, 3]

myFilter :: (a → Bool) → [a] → [a]

myFilter [] = []

myFilter f (x:xs) = f(x) :: (myFilter f xs) ← 递归调用

myFilter f (x:xs) = f(x) :: (myFilter f xs) ← 递归调用

otherwise = myFilter f xs

Python - P25

class float(x)

当不传入参数时, float() 返回值 0.0

当传入参数 x 的类型为 Integer 或 Floating point number 时

则返回一个基于 Python 浮点精度的与原值相等的浮点数

with the same value within Python's floating point precision

如 float(1023) → 1023.0

特别的当参数值 outside the range of Python float

则会抛出错误 OverflowError

注意当数值大于约 10^{310} 之后, float() 返回 inf

如 float('6.02e310') → inf

而当数值绝对值小于约 10^{-320} 之后, float() 返回 0.0

如 float('-6.02e-330') → -0.0

当参数 x 的类型为字符串时, float() 根据语法将其转换为浮点值

输入字符串应包含一个十进制数字, 可以带符号 '+' 或 '-' , '+' 不影响数值

字符串的首尾可以包含空白符 (white space), 如 '\n', '\r', '\t', ' '(空格)

如 float('\t\r 6.02e23 \n') → 6.02e+23

字符串中的十进制数也可以替换为特殊字符串, 并且对大小写不敏感

如 'NaN' (Not a Number), 'inf'/'infinity'

float('n\r n\t n\t n') → nan, float('t-iNf\nn') → -inf

在首尾的空白符去除后, 参数字符串必须符合语法

<sign> ::= "+" | "-"

<infinity> ::= "Infinity" | "inf"] 注意在实现中这两部分对大小写不敏感

<nan> ::= "nan"] 注意在实现中这两部分对大小写不敏感

<numeric_value> ::= <floatnumber> | <infinity> | <nan>

start → <numeric_string> ::= [<sign>] <numeric_value>] 注意此处 [] 表示可选

遵循

Floating Point
literal

其实现允许范围
是依据实现决定的
Implementation
dependent

<floatnumber> ::= <pointfloat> | <exponentfloat>

<exponentfloat> ::= (<digitpart> | <pointfloat>) <exponent>] 即科学计数法

<pointfloat> ::= [<digitpart>] <fraction> | <digitpart> ". "

<fraction> ::= ". " <digitpart>] 注意 . 内通常是多选一

<exponent> ::= ("e" | "E") ["+" | "-"] <digitpart>] 注意 () 内通常是多选一

<digitpart> ::= <digit> (["-"] <digit>)*] float() 支持用下划线 "_" 划分数字分组

general () * 表示 0 个或更多

注意, 更一般的 float(x), 其中 x 是 Python object, 则等价于 x.__float__()

Python - P26

format (value[, format_spec]), 根据传入的 format_spec, 将 value 转换成指定格式
convert value to a "formatted" representation, as controlled by format_spec

format-spec 的表示依赖于 value 参数的类型

当 format-spec 为空字符串时, format() 函数等同于 str()

即 format(123, '') → '123', str(123) → '123'

对于 format(value, format-spec) 的调用

会转换为 type(value).__format__(value, format-spec)

即 Python 会查找实例 value 所属类中的 __format__() 方法

特别地有, 当查找到方法为 object.__format__() 且 format-spec 不为空时

会抛出 TypeError 错误

而当 format-spec 本身或其表达式的返回值不是 string 类型时
也会抛出 TypeError 错误

直接看在 str.format() 方法中, 格式化的 string 中包含需要格式化的部分

称为 replacement field, 在格式化字符串中以 {...} 的形式,

注意格式化字符串中如果需要保留字面上的 {...}, 则需表示为 {{...}}

在 Format String Syntax 中给出了 replacement field 的语法定义

<replacement-field> ::= "{" [<field-name>] ["!" <conversion>] [":" <format-spec>]

<field-name> ::= <arg-name> ("." <attribute-name>) | "[" <element-index> "]"

<arg-name> ::= [<identifier> | (<digit>)+]

<attribute-name> ::= <identifiers>

<element-index> ::= (<digit>)+ | <index-string>

<index-string> ::= (<any character except "}">)*

<conversion> ::= "r" | "s" | "a" | <substitution>

<format-spec> ::= [[<fill>]<align>][<sign>][#"0"] [<width>][<grouping-option>]

[<fill> | <align> | <sign> | # | 0 | <width> | <grouping-option>]

<fill> ::= <any character>

<align> ::= "<" | ">" | "=" | "^"

<sign> ::= "+" | "-"

<width> ::= (<digit>)+

<grouping-option> ::= "-" | ","

<precision> ::= (<digit>)+

<type> ::= "b" | "c" | "d" | "e" | "E" | "f" | "F" | "g" | "G" | "n" | "o" | "s" | "x" | "X" | "%"

Python - P27

str.format() 在 Python 中用 str.format() 对字符串进行格式化
通常调用方法为 '`...{replacement field}...`'.format(value)
其中用 {} 括起来的部分称为 replacement field, 即 <replacement-field>
而字符串的其他部分则看作一般文本 (literal text),
在 format() 过程中, literal text 不作修改而直接输出
如果需要在 literal text 中使用 {}, 则需写为双层括号 {{...}}

replacement field 分为三个可选部分, field name, conversion field, specification

通常来说, <field-name> 的第一部分为 数字 或 关键字

当 <arg-name> 为 数字 (<digit>)⁺ 时

调用传入 str.format() 的 位置参数 (position argument)

如 '{0}'.format('a', 'b') → 'a'

当 <arg-name> 为 关键字 <identifier> 时

调用传入 str.format() 的 关键字参数 (keyword argument)

如 '{b}'.format(a=1, b=2) → '2'

注意, 由于 <arg-name> 不是 quote-delimited 的,

所以 <arg-name> 无法接受任意的关键字 (arbitrary dictionary key), 如 '10'

当 <arg-name> 使用传入的列表、元组、字典等可以 通过索引取值 的参数

可以通过加上 "[<element-index>]" 查值,

如 '{0[1]}'.format([1, 2, 3]) → '2'

'{0[name]}'.format({'name': 'abc', 'score': 66}) → 'abc'

理论上 [index] 的形式可以应用于任何提供 __getitem__() 的对象

当 <arg-name> 传入的对象有 可调用 的属性, 如 complex().real

可以通过加上 ".<attribute-name>" 调用, 使用 getattr() 函数

如 class abc: '{0.a}, {0.b}, {0.c}'.format(abc())

a, b, c = 1, 2, 3 → '1, 2, 3'

注意, 这个方法只能调用成员属性而无法调用成员函数

当 <arg-name> 为空 (omitted) 时, format() 会按顺序分配已传入的位置参数

如 '{}, {}, {}'.format(*[1, 2, 3]) → '1, 2, 3'

这种形式可以和 关键字 混用, 并且顺序分配只考虑 {}

如 '{}, {}, {}'.format(*[1, 2, 3], a=11) → '1, 11, 2'

但是注意这种形式不能和 指定位置参数 混用

如 '{}{}, {}, {}'.format(*[1, 2, 3]), 才抛出错误

cannot switch from automatic field numbering to manual field specification

Python - P28

`str.format()` 在 replacement field 中 "`!<conversion>`" 可选地表示该部分使用的转换方法
其中 `<conversion>` 有 3 种选择 "`r`", "`s`", "`a`".
"r" 表示对 value 调用 `repr()`, 如 `'{:0!r}'.format(...)`
"s" 表示对 value 调用 `str()`, 如 `'{:name!s}'.format(...)`
"a" 表示对 value 调用 `ascii()`, 如 `'{:!a}'.format(...)`

在 replacement field 中 "`:`" `<format-spec>` 可选地表示如何表达 value.

其中 `<format-spec>` 依据 format specification mini-language

并且 `<format-spec>` 中可以嵌套地包含 `<replacement-field>`

这个 replacement field 可以包含 `<field-name>`, `<conversion>`, `<format-spec>`

但是不能存在更深层次的嵌套

另外这个 replacement field 的表达是先于包含它的 `<format-spec>` 的

于是允许 `<format-spec>` 可以动态地定义 (dynamically specified)

迷你格式规范语言 (format specification mini-language), 在 Python 中用于定义 `<format-spec>`

基本语法结构为 `<format-spec> ::=`

`[<fill>]<align> [<sign>] ['#'] ['0'] [<width>][<grouping-option>][.<precision>]`

`<width>` 是一个十进制整型值, 可选地表示 replacement field 的最小宽度 (minimum)
如果 `<width>` 忽略, 则 field 的宽度依据实际情况决定

当采用 "`0`" `<width>` 时, 对于输入的 value 为数字时

在数字的总长度不足 `<width>` 时, 在数字前填充 0

如 `'{:02}', '{:03}', '{:04}'`.format(111, 1, 1.1) → '111, 001, 01.1'

注意当 value 为复数 `complex()` 时, 会出现错误

zero padding is not allowed in complex format specifier

`<grouping-option>` 可选地表示如何对数字进行分隔, 即 thousands separator

可以选择 `,`, 对十进制整数进行划分

如 `'{:0:,3}'.format(2000000)` → '2,000,000'

也可以选择下划线 '`_`', 对十进制划分与 '`,`' 类似, 但可用于二进制, 八进制, 和十六进制

对于二, 八, 十六进制, 按每 4 位数字进行划分

`'{:0:_b}', '{:0:_o}', '{:0:_d}', '{:0:_x}'`.format(1000000)

→ '1111_0100_0010_0100_0000', 364_1100, 1,000,000, f_4240'

Python - P29

2019-11-4

`str.format()` 在 Python 的 format specification mini-language 中

`<sign>` 可选地表示对于数值类型为数字的 value，如何表示正负号

有 "+"、"-"、"(space) 三种选择，如对于 value 为 3.14 和 -3.14

"+" 表示强制注明正号与负号，即有 '+3.14'，'-3.14'

"-" 表示只注明负号，而省略正号，即有 '3.14'，'-3.14'

" " 表示负号如常，而正号以空格代替，即有 ' 3.14'，'-3.14'

`[<fill>]<align>` 可选地表示在给定的宽度中，如何排列字符，以及可选地表示如何填充

有 "<"、"^"、">"、"=" 四种选择，其中 "=" 只针对数字类型的 value

"<"、"^"、">" 分别表示左对齐、居中、右对齐，

空的位置以指定的 `<fill>` 字符填充，如果忽略则以空格 " " 填充

"=" 表示在数字的符号与数字本身之间填充指定字符

注意 "0" `<width>` 实际上被解释为 "0=" (`<fill><align>`) 的格式

如 '{0:*=+7}'，'{0:0=+7}'，'{0:+07}' .format(3.14) → '+**3.14, +003.14, +003.14'

`<type>` 表示对于传入的 value，以何种类型进行表达

available string ["s" 表示字符串格式 (string format)，是字符串的默认类型]

presentation (None) 通常会被忽略，默认的选择也是 "s"

available integer ["b", "o", "d" 表示将整型值表示为二进制数字、八进制数字或十进制数字]

integer presentation ["x", "X" 表示将整型值表示为十六进制数字，"X" 表示用 A-F 代替 a-f]

"c" 表示将整型值转换为对应的 unicode 字符

(None) 忽略时默认的选择是 "d"

integer floating presentation ["e", "E" 表示用科学计数法表示浮点数，"E" 表示用 E 代替 e (scientific notation)]

"f", "F" 表示用定点格式 (fixed-point notation) 表示浮点数，"F" 表示用 'NAN' 'inf' 'nan' 'inf'

"g", "G" 表示用通常格式 (general format) 表示浮点数，(None) 时的默认选择

主要区别在于 "g" 中会在小数部分为 0 时忽略

"%" 表示用百分比形式表示浮点数，使用 "s" 并在最后加上 "%"

指定的分隔符

特殊 ["n" 当整型值时等同于 "d"，浮点时等同于 "g"，使用当前环境 (current locale setting)]

"." `<precision>` 可选地表示保留小数位数 ("f")，保留有效数字 ("g")

当类型为字符串时，表示保留最大长度 ("s")，不可用于整型值

如 <precision>

如 <precision>

"#" 可选地表示替代形式 (alternative form)，对于不同的 `<type>` 有不同效果

如对于整型值在前端加上 '0b', '0o', '0x'