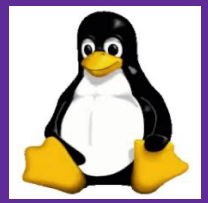
Introduction to Computation

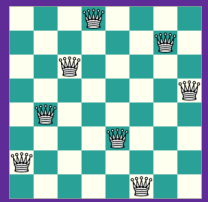
Autumn, 2023

Prof. Fan Cheng

Shanghai Jiao Tong University

chengfan85@gmail.com
https://github.com/ichengfan/itc





14

Outline

- Functional Programming
- LEGB

Functional Programming

函数式编程

语义辨析：statement Vs. expression

- Expressions only contain identifiers, literals and operators, where operators include arithmetic and Boolean operators, the function call operator () the subscription operator [] and similar, and can be reduced to some kind of "value", which can be any Python object
  - 2*2, x and y or z, (x, y)[z>w]
  - is_prime(123) # return True or False
- Statements, on the other hand, are everything that can make up a line (or several lines) of Python code. Note that expressions are statements as well
  - x = 123
  - def func():, class Test():
- An expression evaluates to a value. A statement does something
  - Expressions produce at least one value

Literals

Literals are notations for constant values of some built-in types.

https://docs.python.org/3/reference/lexical_analysis.html#literals

https://www.programiz.com/python-programming/variables-constants-literals

:=

There is new syntax := that assigns values to variables as part of a larger expression. It is affectionately known as “the walrus operator” due to its resemblance to the eyes and tusks of a walrus.

```
1   x = 1
2   y = x + 1
3   print(y)
4
5   # y = (x = 1) + 1 # Error
6
7   y = (x := 1 + 1)
8   print(y)
9
10  # x := 7 # Error
11  (x := 7)
12  print(x)
```

```
2
2
7
```

Lambda： 匿名函数

- A lambda function is a small anonymous function
- A lambda function can take any number of arguments, but can **only have one expression**
- **The return value is the value of the expression**
- 关键字 **lambda**

  lambda arguments : expression

高阶函数

```
# Add 10 to argument a, and return the result:
x = lambda a : a + 10
print(x(5))

# Multiply argument a with argument b and return the result:
x = lambda a, b : a * b
print(x(5, 6))

# Summarize argument a, b, and c and return the result:
x = lambda a, b, c : a + b + c
print(x(5, 6, 2))

def myfunc(n):
  return lambda a : a * n

mydoubler = myfunc(2)
mytripler = myfunc(3)

print(mydoubler(11))
print(mytripler(11))
```

```
15
30
13
22
33
```

```
f = lambda lst, x: lst.append(x) or lst
lst = list(range(11))
print(f(lst, -12))
```

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, -12]
```

Lambda: sorted

- The sorted() function returns a sorted list of the specified iterable object
  - List.sort(), sort in place, return None
  - https://docs.python.org/3/howto/sorting.html

```
1  print(sorted([1, 2, 3, -1, 10, -5]))
2  print([1, 2, 3, -1].sort())
3  print(sorted({1, 2, 3, -1, 10, -5}))
4  print(sorted((1, 2, 3, -1, 10, -5)))
```

```
[-5, -1, 1, 2, 3, 10]
None
[-5, -1, 1, 2, 3, 10]
[-5, -1, 1, 2, 3, 10]
```

- Parameter:
  - key Optional. A Function to execute to decide the order. Default is None
  - reverse Optional. A Boolean. False will sort ascending, True will sort descending. Default is False
- Key: Based on the returned value of the key function, you can sort the given iterable.
  - The function may be lambda, self-defined function
  - Operator: https://docs.python.org/3/library/operator.html#module-operator
- In python2, cmp may be used to compare two elements and sort them
- CMP is removed in Python 3. You must implement >=, <=, >, <, ==, != (rich comparison)

https://py.checkio.org/blog/how-did-python3-lose-cmd-sorted/

```python
print("sorted by abs()")
print(sorted([1, 2, 3, -1, 10, -5], key=abs))
print(sorted({1, 2, 3, -1, 10, -5}, key=abs))
print(sorted((1, 2, 3, -1, 10, -5), key=abs))


def pow2(x):
    return x**2


print("sorted by pow2()")
print(sorted([1, 2, 3, -1, 10, -5], key=pow2))
print(sorted({1, 2, 3, -1, 10, -5}, key=pow2))
print(sorted((1, 2, 3, -1, 10, -5), key=pow2))


def f(x):
    return x**2 - 10 * x


print("sorted by f()")
print(sorted([1, 2, 3, -1, 10, -5], key=f))
print(sorted({1, 2, 3, -1, 10, -5}, key=f))
print(sorted((1, 2, 3, -1, 10, -5), key=f))

print("sorted by lambda")
print(sorted([1, 2, 3, -1, 10, -5], key=lambda x: abs(x)))
print(sorted([1, 2, 3, -1, 10, -5], key=lambda x: x**2))
print(sorted([1, 2, 3, -1, 10, -5], key=lambda x: x**2 - 10 * x))
```

```
sorted by abs()
[1, -1, 2, 3, -5, 10]
[1, -1, 2, 3, -5, 10]
[1, -1, 2, 3, -5, 10]
sorted by pow2()
[1, -1, 2, 3, -5, 10]
[1, -1, 2, 3, -5, 10]
[1, -1, 2, 3, -5, 10]
sorted by f()
[3, 2, 1, 10, -1, -5]
[3, 2, 1, 10, -1, -5]
[3, 2, 1, 10, -1, -5]
sorted by lambda
[1, -1, 2, 3, -5, 10]
[1, -1, 2, 3, -5, 10]
[3, 2, 1, 10, -1, -5]
```

Lambda + dict

- A lambda expression could be the value of a dict

  dict[key]=lambda x, y: x + y
- An alternative to if … elif…elif…else statement

```
dt = {
    "1": (lambda x, y: x + y),
    "2": (lambda x, y: x - y),
    "3": (lambda x, y: x * y),
    "4": (lambda x, y: x**y),
}

keylist = ["1", "3", "2", "4", "1", "2"]
for x in keylist:
    print(x, dt[x](10, 3))
```

```
1 13
3 30
2 7
4 1000
1 13
2 7
```

Lambda: 蚂蚁虽小

仅供了解

- A lambda function can take any number of arguments, but can **only have one expression**
- Expression VS. Statement: Expressions should have value
  - Lambda cannot contain print(), “=“ assignment, etc
  - Python 3.8, := assignment
- Lambda is almost as power as normal python statement: If you know what you’re doing, though, you can code most statements in Python as expression-based equivalents.
- if a: b else: c的N种写法
  - b if a else c
  - (c, b)[a]
  - ((a and b) or c)

```
def max1(a, b):
    if a>=b:
        return a
    else:
        return b

def max2(a, b):
    return a if a>=b else b

def max3(a, b):
    return (a, b)[a<b]

def max4(a, b):
    return ((a>=b) and a) or b

print(max1(1, 2), max2(1, 2), max3(1, 2), max4(1, 2))
print(max1(2, 2), max2(2, 2), max3(2, 2), max4(2, 2))
print(max1(3, 2), max2(3, 2), max3(3, 2), max4(3, 2))
```

```
2 2 2 2
2 2 2 2
3 3 3 3
```

map 映射

相当于Loop

map(function, iterable, ...): for x in iterable: f(x)

- Return an iterator that applies function to every item of iterable, yielding the results.

```python
def addition(n):
    return n + n

# We double all numbers using map()
numbers = (1, 2, 3, 4)
result = map(addition, numbers)
print(list(result))

M = [[1, 2, 3], # A 3 × 3 matrix, as nested lists
[4, 5, 6], # Code can span lines if bracketed
[7, 8, 9]]
print(list(map(sum, M)))

data = (-1, 0, 1)
M = map(abs, data)
print(data)
print(list(M))
print(data)

def add1(x):
    return x+1

add = map(add1, data)
print(list(add))
```

```
[2, 4, 6, 8]
[6, 15, 24]
(-1, 0, 1)
[1, 0, 1]
(-1, 0, 1)
[0, 1, 2]
```

filter 过滤

filter(function, iterable): select the element x where function(x) is True

- Construct an iterator from those elements of iterable for which function returns true. iterable may be either a sequence, a container which supports iteration, or an iterator. If function is None, the identity function is assumed, that is, all elements of iterable that are false are removed.

```
f = filter(bool, ['spam', '', 'ni']) # bool(x)
print(list(f))

def ff(x):
    return x**2 - 4*x >= 0

f = filter(ff, range(-10,11))
print(ff)
print(list(f))
```

```
['spam', 'ni']
<function ff at 0x000002231E86C700>
[-10, -9, -8, -7, -6, -5, -4, -3, -2, -1, 0, 4, 5, 6, 7, 8, 9, 10]
```

Use reduce() to merge two dicts

reduce: basic

- Map-reduce
- 从头到尾迭代：sum

- The reduce(fun, iterable) function is used to apply a particular function passed in its argument to all of the list elements mentioned in the sequence passed along
- This function is defined in “functools” module

```
from functools import reduce # Import in 3.X, not in 2.X
def my_add(a, b):
    result = a + b
    print(f"{a} + {b} = {result}")
    return result

numbers = [0, 1, 2, 3, 4]
print(reduce(my_add, numbers))

print(reduce((lambda x, y: x + y), [1, 2, 3, 4]))

print(reduce((lambda x, y: x * y), [1, 2, 3, 4]))
lis = [ 1 , 3, 5, 6, 2, ]

# using reduce to compute sum of list
import functools
print ("The sum of the list elements is : ",end="")
print (functools.reduce(lambda a,b : a+b, lis))

# using reduce to compute maximum element from list
print ("The maximum element of the list is : ", end="")
print (functools.reduce(lambda a,b : a if a > b else b, lis))
```

```
0 + 1 = 1
1 + 2 = 3
3 + 3 = 6
6 + 4 = 10
10
10
24
The sum of the list elements is : 17
The maximum element of the list is : 6
```

```
import operator, functools
print(functools.reduce(operator.add, [2, 4, 6])) # Function-based +
print(functools.reduce((lambda x, y: x + y), [2, 4, 6]))
```

```
12
12
```

reduce: initializer

- 从头到尾迭代：sum

- reduce(fun, iter, initial)
- reduce的工作流程是：
  - 有initial参数的情况：
    $y_1 = f(x_0, x_1), y_2 = f(y_1, x_2), \dots, f_n = f(y_{n-1}, x_n)$
  - 没有initial参数的情况：
    $y_1 = f(x_1, x_2), y_2 = f(y_1, x_3), \dots, f_n = f(y_{n-1}, x_n)$

```python
lis = list(range(10))
print(reduce(lambda x, y: x + y, lis))

lis = list(range(10))
print(reduce(lambda x, y: x + y, lis, -100))

lis = list(range(2, 10)) # 1^2+2^2+...+9^2
print(reduce(lambda x, y: x + y**2, lis, 0))

lis = list(range(2, 10)) # ???
print(reduce(lambda x, y: x + y**2, lis))

lis = list(range(2, 10)) # ???
print(reduce(lambda x, y: x**2 + y**2, lis))
```

```
45
-55
284
282
3585661762209874419959583726903982869987246161688487820032323950038010002
```

Example 1:

- 删除一个例表中重复元素，并保留原有顺序

```python
lis = [-1, 1, -2, 2, -3, 5, 4, 6, 3, 8, -1, 8, -2, -3, 1, 2, 3]


def flist(x, y):
    st, li = x
    if y not in st:
        st.add(y)
        li.append(y)
    return (st, li)


print(reduce(flist, lis, (set(), [])))

lis = [-1, 1, -2, 2, -3, 5, 4, 6, 3, 8, -1, 8, -2, -3, 1, 2, 3]
print(
    reduce(
        lambda x, y: y not in x[0] and (x[0].add(y) or x[0], x[1] + [y]) or x,
        lis,
        (set(), []),
    )
)
```

```
({1, 2, 3, 4, 5, 6, 8, -2, -3, -1}, [-1, 1, -2, 2, -3, 5, 4, 6, 3, 8])
({1, 2, 3, 4, 5, 6, 8, -2, -3, -1}, [-1, 1, -2, 2, -3, 5, 4, 6, 3, 8])
```

Example 2:

- 找出一个集合中第二小的元素

```
num_list = [1, 2, 4, 33, 6, 22, 9, 13]
num = reduce(
    lambda ot, x: ot[1] < x and (ot[1], x) or ot[0] < x and (x, ot[1]) or ot,
    num_list,
    (0, 0),
)[0]
print("Second_large_num is :", num)
```

```
Second_large_num is : 22
```

Example 3:

- 判断一个数是不是happy number：即各个位的平方和最后变成1

```python
def is_happy(n):
    st = set()
    while n != 1:
        lis = [int(x) for x in str(n)]
        rn = reduce(lambda x, y: x + y**2, lis, 0)
        if rn in st:
            return False
        st.add(rn)
        n = rn
    return n == 1


not_happy = lambda n: not is_happy(n)

ans = list(filter(lambda n: not is_happy(n), range(1, 10001)))
print(len(ans), ans[0], ans[1], ans[-1])

ans = list(filter(not_happy, range(1, 10001)))
print(len(ans), ans[0], ans[1], ans[-1])
```

```
8558 2 3 9999
8558 2 3 9999
```

Functional Programming Modules

- The modules described in this chapter provide functions and classes that support a functional programming style, and general operations on callables
- The following modules are documented in this chapter:
- itertools — Functions creating iterators for efficient looping
  - Itertool functions
  - Itertools Recipes
- functools — Higher-order functions and operations on callable objects
  - partial Objects
- operator — Standard operators as functions
  - Mapping Operators to Functions
  - In-place Operators

https://docs.python.org/3/library/functional.html

怀旧篇: C, Java

- printf
- system.out

```python
import functools

printf = functools.partial(print, end="")


class Empty:
    pass


System = Empty()
System.out = Empty()
System.out.println = print


printf("hello world\n")
System.out.println("hello world")
```

Programming Paradigm

- Imperative
  - 函数变，数据变
- Object
  - 函数不变，数据变
- Functional: Lisp, Scala, Haskell
  - 函数变，数据不变

Reading

- SICP
- Learning Python
- 不要滥用map, filter, reduce, list comprehension
- 清晰，可读，正确，可维护
- Python is lisp

LEGB

作用域

变量的作用域

一个变量名可能存在的区域：LEGB

- B—系统内置的变量，譬如pow, print, id等等，称为built-in
- G—模块里面的全局变量，称为global
- L—单个函数内部变量，称为local
- E—封闭区域(嵌套函数、类定义)内部的变量，称为enclosing

Python的设计原则

- 不同区域的变量互不干扰，互不影响
- 同一个区域，一个变量只能属于LEGB中一种，只有一个起作用
- 变量在一个区域的LEGB属性保持不变

```
def outer():
    x = 123

    def inter():
        print(x)

    inter()

outer()
```

x属于一个封闭区域(Enclosing)，但是既不global也不local

Enclosing

- x: global. x3: local
- x1, x2: enclosing
  - 在一个密封环境中, def内部
  - 相对于x是local
  - 相对于x3是global，可以被函数f2, f3使用

```
x = -1000
def f1():
    x1 = 100
    print(x, x1)

    def f2():
        x2 = 101
        print(x, x1, x2)

        def f3():
            x3 = 102
            print(x, x1, x2, x3)

        f3()

    f2()

f1()
```

```
-1000 100
-1000 100 101
-1000 100 101 102
```

```
def f(n):
    return lambda : sum(range(n+1))
print(f(100)())
```

```
5050
```

Namespace 名字空间

Namespaces : A namespace is a container where names are mapped to objects, they are used to avoid confusions in cases where same names exist in different namespaces. They are created by modules, functions, classes etc.

- 名空间中：同一个范围的名字所构成的空间
  - 外部变量也会在空间产生作用，但是不属于这个空间
  - **名空间由变量定义唯一决定**
- 一个相同名字的变量，可能出现在不同的名空间，也就有不同的**作用域**
- 名空间local、global、enclosing、built-in
  - local：函数内部
  - global：模块顶层
  - 嵌套函数怎么算: Enclosing
  - built-in 系统自定义的：pow, print, sum等等
- python中变量必须先定义再使用
  - 赋值语句variable = value
  - 预示新的变量及其作用域的诞生
  - 使用一个变量不会改变其作用域

```
def f():
    print(s)
s = "I love Paris in the summer!"
f()
```

global

```
def f():
    s = "I love London!"
    print(s)

s = "I love Paris!"
f()
print(s)
```

Global and Local

```
def f():
    s = 100
    s += 200
    print(s)
    del s
    print(s)

s = 100
f()
```

error

= always determine name scopes unambiguously

global

- 在全局范围以外的区域修改一个全局变量，需要提前声明其为global
- 同一个作用域，只有一个同名变量起作用

```
def f():
    s += 200
    print(s)


s = 100
f()
```

UnboundLocalError: local variable 's' referenced before assignment

```
def f():
    global s
    s += 200
    print(s)


s = 100
f()
```

```
300
```

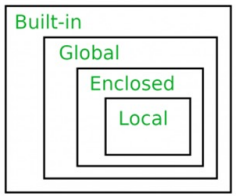
```
def f():
    s = 100
    s += 200
    print(s)


s = 100
f()
```

```
300
```

f()内部的s和外部的s是两个不同的变量

Namespace: LEGB Rule

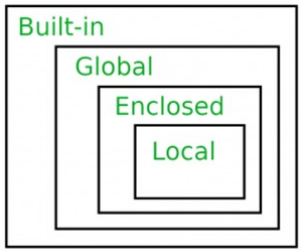
- Python’s name-resolution scheme is sometimes called the LEGB rule
- When you use an unqualified name inside a function, Python searches up to four scopes—the local (L) scope, then the local scopes of any enclosing (E) defs and lambdas, then the global (G) scope, and then the built-in (B) scope—and stops at **the first place the name is found**
  - If the name is not found during this search, Python reports an error



1. 变量赋值决定名空间
2. LEGB原则
3. 同一个空间中，一个名字只有唯一一个LEGB属性

- Local: When you assign a name in a function (instead of just referring to it in an expression), **Python always creates or changes** the name in the local scope, unless it’s declared to be global or nonlocal in that function
- Global: When you assign a name outside any function (i.e., at the top level of a module file, or at the interactive prompt), the local scope is the same as the global scope — the module’s namespace.

LEGB：从内到外，从小到大各占一方、互不关涉



1. 变量赋值决定名空间
2. LEGB原则
3. 同一个空间中，一个名字只有唯一一个LEGB属性

- L-Local (function)：函数内的名字空间
  E-Enclosing function locals：外部def的名字空间(例如类，嵌套函数)
  G-Global (module)：函数定义所在模块（文件）的名字空间
  B-Builtin (Python): Python内置模块的名字空间

```
def f():
    print(s)
s = "I love Paris in the summer!"
f()
```

```
def f():
    s = "I love London!"
    print(s)

s = "I love Paris!"
f()
print(s)
```

```
def f():
    print(s)
    s = "I love London!"
    print(s)

s = "I love Paris!"
f()
```

```
def f():
    global s
    print(s)
    s = "Only in spring, but London is great as well!"
    print(s)


s = "I am looking for a course in Paris!"
f()
print(s)
```

```
def f():
    s = "I am globally not known"
    print(s)

f()
print(s)
```

```python
def foo(x, y):
    global a
    a = 42
    x, y = y, x
    b = 33
    b = 17
    c = 100
    print(a, b, x, y)


a, b, x, y = 1, 15, 3, 4
foo(17, 4)
print(a, b, x, y)
```

```python
def f():
    city = "Hamburg"
    def g():
        global city
        city = "Geneva"
    print("Before calling g: " + city)
    print("Calling g now:")
    g()
    print("After calling g: " + city)

f()
print("Value of city in main: " + city)
```

```python
x = -1000
def f1():
    x1 = 100
    print(x, x1)

    def f2():
        x2 = 101
        print(x, x1, x2)

        def f3():
            x3 = 102
            print(x, x1, x2, x3)

        f3()

    f2()

f1()
```

```python
for x in range(100):
    for y in range(100):
        pass

print(x, y)
```

nonlocal

一个变量，既不是local也不是global

- The nonlocal keyword is used to work with variables inside nested functions, where the variable should not belong to the inner function
- Use the keyword nonlocal to declare that the variable is not local
- Global, nonlocal, local: three status, only one should exist in a program

```
def myfunc1():
  x = "John"
  def myfunc2():
    nonlocal x
    x = "hello"
  myfunc2()
  return x

print(myfunc1())
```

```
hello
```

```
def f():
    city = "Munich"
    def g():
        nonlocal city
        city = "Zurich"
    print("Before calling g: " + city)
    print("Calling g now:")
    g()
    print("After calling g: " + city)

city = "Stuttgart"
f()
print("'city' in main: " + city)
```

```
Before calling g: Munich
Calling g now:
After calling g: Zurich
'city' in main: Stuttgart
```

```
def f():
    #city = "Munich"
    def g():
        nonlocal city
        city = "Zurich"
    print("Before calling g: " + city)
    print("Calling g now:")
    g()
    print("After calling g: " + city)

city = "Stuttgart"
f()
print("'city' in main: " + city)
```

```python
x = "g1"
def g1():

    def g2():
#       nonlocal x
        global x
        print("g2. {}".format(x))
        x = "g2"
        print("g2. {}".format(x))


    global x

    print(x)
    g2()
    print(x)
    x = "gg"
    print(x)

g1()
```

Why nonlocal is wrong?