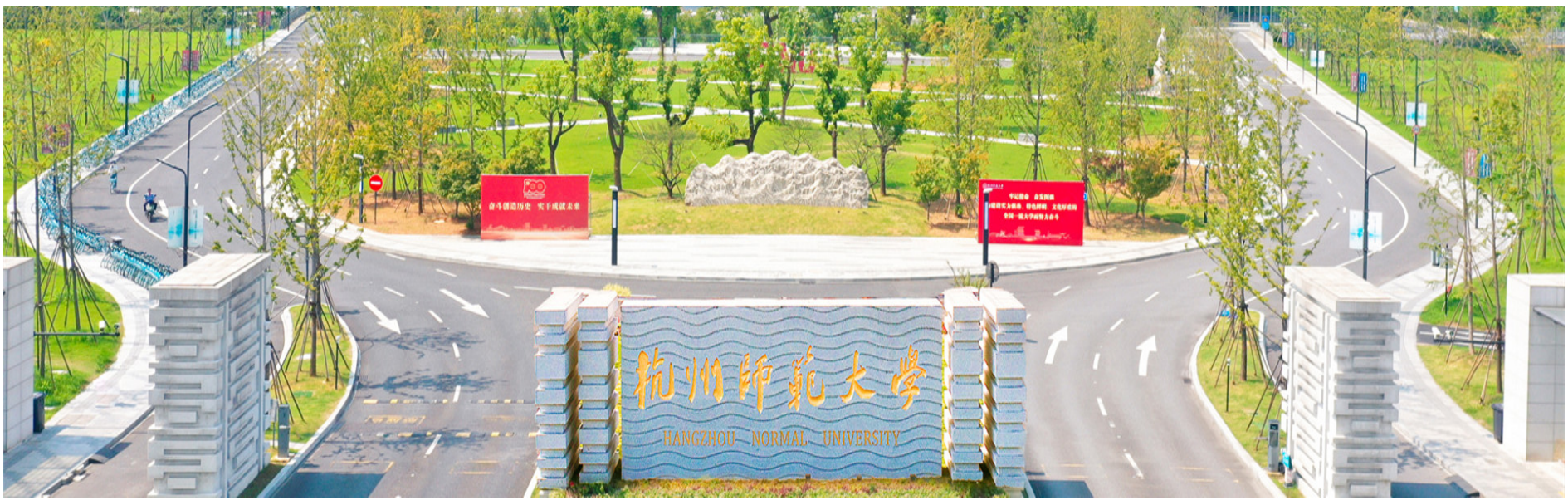
第五讲 - Python 函数

张建章

阿里巴巴商学院

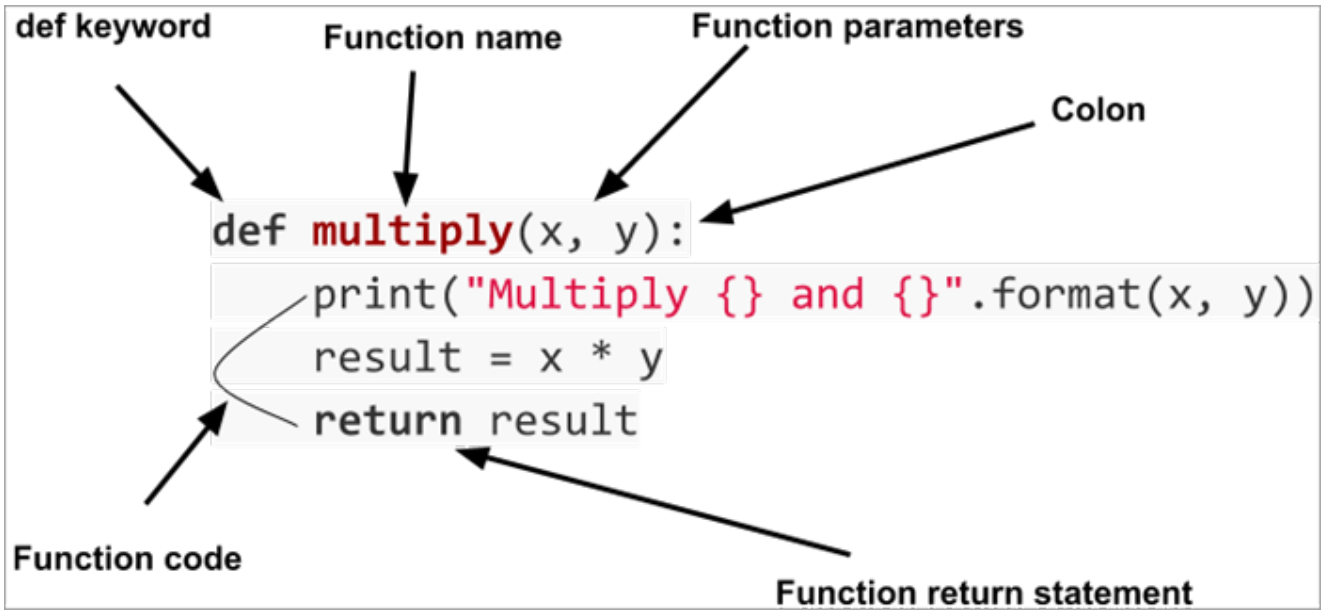
杭州师范大学

2022-09



目录 I

函数：为实现一个操作或特定功能而集合在一起的语句集，避免代码复制带来的错误或漏洞，不仅可以实现代码的复用，还可以保证代码的一致性。



一个完整的函数由如上图所示的几部分组成：`def` 关键字 (def keyword)，函数名 (function name)，函数参数 (function parameters)，冒号 `:` (colon)，函数体 (function code)。函数体内 `return` 语句只执行一次，执行完成后，不再运行后续语句。

```
def multiply(x,y):
  print("Multiply {} and {}".format(x, y))
  result = x * y
  return result
```

注意：函数体中的`return`语句不是必需的，如果不写，调用函数后返回`None`，如果所定义的函数不需要参数，可以不写参数，但是括号必需写，看下面例子：

```
def color_print():
  for i in range(91,96):
    print("\033["+str(i)+"m"+"zjzhang"+"\033[0m")

result = color_print()
if result == None:
  print('The call of Function color_print returns None')
```

函数调用

```
# 调用multiply函数，忽略返回值
multiply(3, 5)

# 调用multiply函数，并将函数的返回值复制给变量r
r = multiply(3, 5)

print(r) # 15

# 调用color_print函数，忽略返回值None
color_print()

# 调用color_print函数，将返回值赋值给变量r
# 对于没有参数的函数，这种调用方式没什么意义
r = color_print()
print(r)
```

注意: ① 函数调用可以将返回值赋值给其他变量，也可以忽略返回值；② 函数调用必须写括号，无论是否需要写参数。

局部变量: 在函数体内定义的变量，只能在函数内部被访问。

全局变量: 在所有函数之外创建的变量，可以在程序的任意位置访问。

```
# 此处的x, y为全局变量
x, y = 3, 5

def myfunct():
  # 此处的x, y为局部变量
  x, y = 8, 9
  result = x * y

p = myfunct() # 函数调用
# 使用全局变量做计算并将计算结果赋值给全局变量z
z = x + y
print(z) # 8, 不是17
print(p) # None
```

由上例可见，局部变量和全局变量可以同名，但是在函数内部定义的局部变量不能在函数外访问。

形式参数与实际参数

```
# 定义函数
def multiply(x,y):
  print("Multiply {} and {}".format(x, y))
  result = x * y
  return result

# 调用函数
p, q = 8, 9
multiply(p, q)
```

在上例中，定义函数时写的参数 x 和 y 是形式参数 (形参)，调用函数时传递给函数的参数 p 和 q (或者说 8 和 9) 叫做实际参数 (实参)。

Python 的参数传递

```python
# 定义函数
def multiply(x,y):
  print("Multiply {} and {}".format(x, y))
  result = x * y
  return result

# 调用函数

p, q = 8, 9
# 这是引用传递，引用了变量p和q的值8和9
multiply(p, q)
# 这是值传递
multiply(5, 6)
```

注意：引用传递的本质是将变量指向的数据对象传递到函数中，上例中，变量 p 和 q 指向的数据对象为整数 8 和 9。

```python
# 定义函数
def my_double(my_obj):
  my_obj *= 2
  return my_obj

a = '000'
b = [1,2]
# 函数调用
my_double(a)
my_double(b)
print(a) # 000
print(b) # [1,2,1,2]
```

上面例子中，变量 a 和变量 b 分别指向不可变对象 (字符串) 和可变对象 (列表)，因此，在函数调用后，变量 b 的值发生了变化，变量 a 的值不变。所以，**当进行引用传递时，要注意，如果变量指向的是可变对象，函数体的代码是可以改变变量值的。**

默认参数

定义函数时，可以为参数指定默认值。

如果在调用函数时，未指定参数，则使用参数的默认值作为参数值运行函数；

如果参数不存在默认值，并且调用时也没指定参数值，则程序出错。

```
def my_power(base, exponent=2):
  return base ** exponent

print(my_power(3)) # 9
print(my_power()) # Error
```

上例中定义的函数 `my_power`，包含两个参数 base 和 exponent，参数 base 没有默认值，在调用函数时必须为其指定值，参数 exponent 有默认值，为 2，调用函数时，如不指定其值，则使用 2 作为其值运行函数。

定义函数时，没有默认值的参数在前，具有默认值的参数在后，否则，程序出错 (SyntaxError)。

```
# Correct
def my_power(base, exponent=2):
  return base ** exponent

# Error
def my_power(exponent=2, base):
  return base ** exponent
```

Python 不支持函数重载。

函数重载 (方法重载): 是某些编程语言 (如 C++、C#、Java、Swift、Kotlin 等) 具有的一项特性，该特性允许创建多个具有不同实现的同名函数。对重载函数的调用会运行其适用于调用上下文的具体实现，即允许一个函数调用根据上下文执行不同的任务。

按位置传参和按关键字传参

按位置传参: 按照函数定义时的顺序进行参数传递。

按关键字传参: 按照 `name=value` 的格式进行参数传递。

```
# my_sum的三个参数均有默认值
def my_sum(a = 1, b = 2, c = 3):
  return a**0 + b**1 + c**2

# 按位置传参
my_sum(7, 8, 9)
# 按关键字传参
my_sum(c=7, a=8, b=9)
# 按位置传参和按关键字传参混合使用
my_sum(7, 8, c=9)

my_sum(7, b=8, 9) # Error
```

注意：按位置传参和按关键字传参混合使用时，位置位置参数在前，关键字参数在后，否则，程序出错 (SyntaxError)。

```
# 按关键字传参，对有默认值的参数使用其默认值
my_sum(a=9)

# 按位置传参，对有默认值的参数使用其默认值
my_sum(7)

# 按位置传参和按关键字传参混合使用
my_sum(7, c=8)

# 按位置和按关键字传参混合使用，并且引用传递和值传递混合使用
x, y, z = 7, 8, 9
my_sum(x, b=z, c=8)

# 按位置传参和按关键字传参混合使用，但是给参数a传递了多个值
my_sum(7, a=8) # Error
```

可变参数

注意：在定义函数时，不固定参数的数量，调用时也可以使用不同个数的参数，可使用两种参数名来实现：`*args` 和 `**kwargs`，分别对应参数元组 (按位置传参) 和参数字典 (按关键字传参)。

```
def hello_args(para1, *args):
  print("para1 :", para1)
  print("type(args):", type(args))
  print("args :", args)
  for idx,arg in enumerate(args):
    print("args{}:".format(idx+1), arg)

hello_args() # Error，必须要为para1指定参数值，因为它没有默认值
hello_args('hello') # 按位置传参，没有为*args传参
hello_args('hello', 'this', 'is', 'mc.zhang') # 按位置传参
```

上例中，`*args` 用来定义个数不确定的参数**元组**，在函数体中可以通过 `for` 循环和索引访问参数元组，对于 `*args`，可以传递 0 或多个参数。

```python
def hello_args(para1, *args):
  print("para1 :", para1)
  print("type(args):", type(args))
  print("args :", args)
  for idx,arg in enumerate(args):
    print("args{}:".format(idx+1), arg)

args_list = ['this', 'is', 'mc.zhang']
# 先序列(列表)解封，再按位置传参
hello_args(*args_list)
# 先序列(列表)解封，再按位置传参
hello_args('hello', *args_list)

# 先序列(列表)解封，再混合使用按关键字传参和按位置传参，不行
# 这样就违背了“位置参数在前，关键字参数在后”的规则
hello_args(para1 = 'hello', *args_list) # Error
```

```python
def hello_kwargs(para1, **kwargs):
  print("para1 :", para1)
  print("type(kwargs):", type(kwargs))
  print("kwargs:", kwargs)
  for key, value in kwargs.items():
    print("{0} = {1}".format(key, value))

hello_kwargs() # Error，必须要为para1指定参数值，因为它没有默认值
hello_kwargs('hello') # 按位置传参，没有为*kwargs传参
# 按位置传参和按关键字传参混合
hello_kwargs("hello", kwarg_1='this', kwarg_2='is',
↪  kwarg_3='mc.zhang')
```

上例中，`*kwargs` 用来定义个数不确定的参数**字典**，在函数体中可以通过 `for` 循环和关键字 (key) 访问参数字典，对于 `*kwargs`，可以传递 0 或多个参数。

```python
def hello_kwargs(para1, **kwargs):
  print("para1 :", para1)
  print("type(kwargs):", type(kwargs))
  print("kwargs:", kwargs)
  for key, value in kwargs.items():
    print("{0} = {1}".format(key, value))

kwargs_dict = {'kwarg_1': 'this', 'kwarg_2': 'is', 'kwarg_3':
↪  'mc.zhang'}
# 先字典解封，再按位置传参和按关键字传参混合
hello_kwargs("hello", **kwargs_dict)
# 按位置传参，Error，对于**kwargs，只能按关键字传参
hello_kwargs('hello', 'this', 'is', 'mc.zhang')
# 先字典解封，再按关键字传参
hello_kwargs(para1 = "hello", **kwargs_dict)
hello_kwargs(**kwargs_dict, para1 = "hello")
# Error，违背了“位置参数在前，关键字参数在后”的规则
hello_kwargs(**kwargs_dict, "hello")
```

可变参数实例

```
def adder(*num):# 求任意个数字之和
  sum = 0
  for n in num:
    sum = sum + n
  print("Sum:",sum)

adder(3,5)
adder(1,2,3,5,6)

def intro(**data):# 打印通讯录
        print("\nData type of argument:",type(data))
        for key, value in data.items():
                print("{} is {}".format(key,value))

intro(Firstname="Sita", Lastname="Sharma", Age=22,
↪  Phone=1234567890)
intro(Firstname="John", Lastname="Wood",
↪  Email="johnwood@nomail.com", Country="Wakanda", Age=25,
↪  Phone=9876543210)
```

```python
def my_max(x,y):
  return x if x > y else y

l = [8,-8]
t = ('good', 'good.')
my_max(*l) # 8
my_max(*t) # 'good.'

def avg_grade(chinese = 80, math = 85):
  return (chinese + math)/2

grade_dict = {'chinese':88, 'math':99}
avg_grade(**grade_dict) # 93.5

def avg_grade_mul(chinese = 80, math = 85, **kwargs):
  return (chinese + math + sum(kwargs.values()))/(2+len(kwargs))

grade_dict = {'english':88,'history':90}
avg_grade_mul(**grade_dict) # 85.75
```

定义函数时不同类型参数的顺序

(位置参数，默认值参数，可变参数元组，命名参数，可变参数字典)

```
def my_func(a, b, c = 0, *args, d, **kwargs):
  print('a:      {}'.format(a),
        'b:      {}'.format(b),
        'c:      {}'.format(c),
        'args:   {}'.format(args),
        'd:      {}'.format(d),
        'kwargs: {}'.format(kwargs),
         sep='\n')

my_func('a', 'b', 88, *[-1, -2, -3], d='d',
↪   **{'name':'Tom','age':'18'})
```

本小节通过一系列函数实例展现函数的功能封装性。

题目：给定一个身份证号，验证其是否合法：① 总体校验；② 校验地区；③ 校验日期。

```python
# 校验字符和长度
def verify_char_length(ids):
  if len(ids) != 18:
    return False
  if not ids[:-1].isdigit():
    return False
  if ids[-1] not in '0123456789X':
    return False
  return True
```

```python
# 校验第18位是否正确
def verify_last_num(ids):
  ids_17 = ids[:-1]
  weights = [7, 9, 10, 5, 8, 4, 2, 1, 6, 3, 7, 9, 10, 5, 8, 4, 2]
  S = sum([int(num)*weight for num,weight in
  ↪  zip(list(ids_17),weights)])
  T = S % 11
  R = (12 - T) % 11
  if R == 10:
    last_num = 'X'
  else:
    last_num = R
  if ids[-1] == str(last_num):
    return True
  else:
    return False
```

```python
# 校验前六位地区编码是否正确
def verify_area(ids):
  import _locale
  _locale._getdefaultlocale = (lambda *args: ['zh_CN', 'utf8'])
  with open('./area_dict.json') as f:
    area_dict = eval(f.read())
  if ids[:6] not in area_dict.keys():
    return False
  else:
    return True

# 校验闰年
def is_leap_year(year):
  if (year % 4 == 0 and year % 100 != 0) or (year % 400 == 0):
    return True
  else:
    return False
```

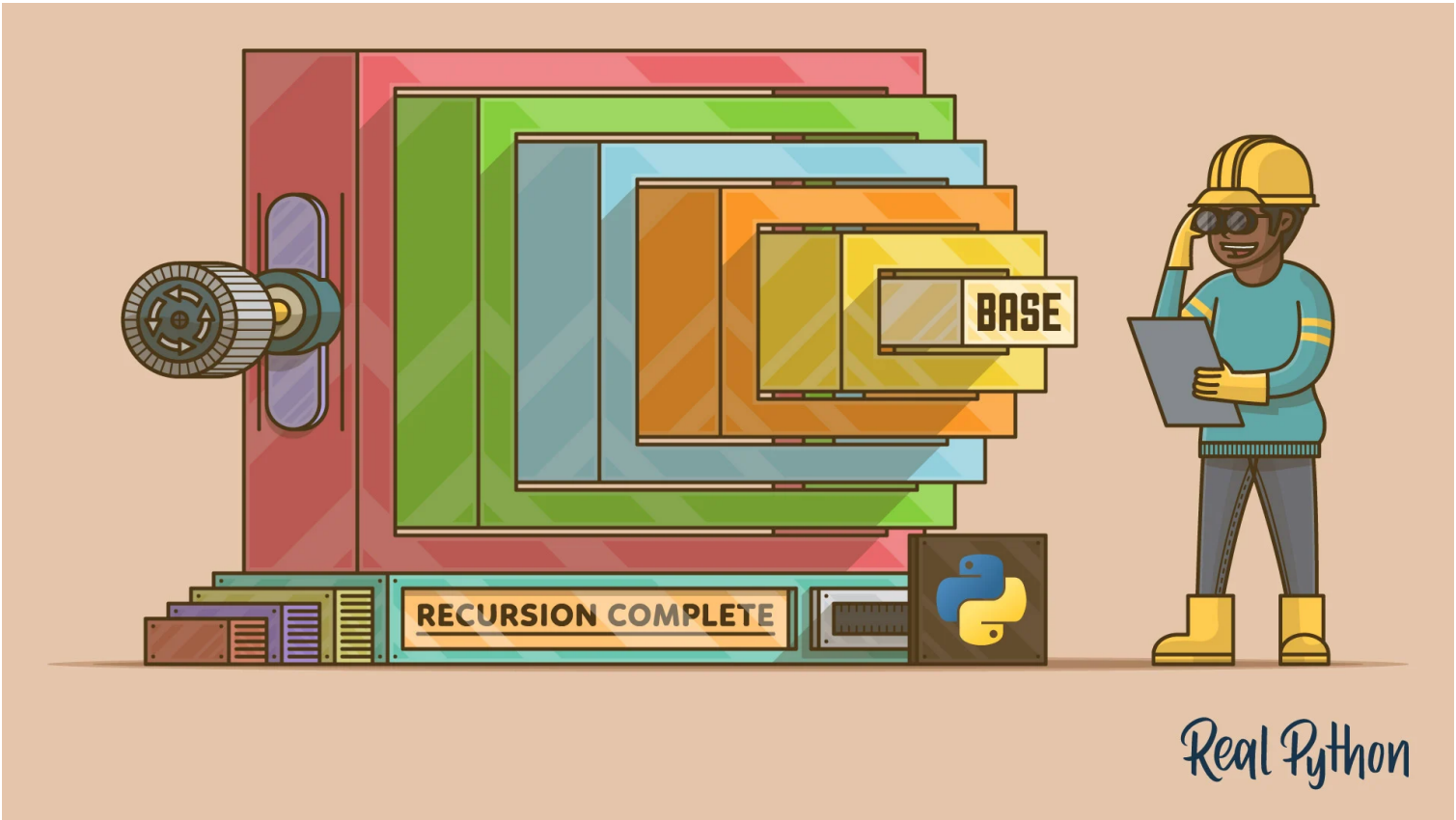
```python
def verify_date(ids):
  year = int(ids[6:10])
  if year > 2022 or year < 1900:
    return False
  month = int(ids[10:12])
  if month > 12 or month < 1:
    return False
  day = int(ids[12:14])
  if month in [1,3,5,7,8,10,12]:
    if day > 31 or day < 1:
      return False
  elif month in [4,6,9,11]:
    if day > 30 or day < 1:
      return False
  else:
    if is_leap_year(year):
      if day > 29 or day < 1:
        return False
    else:
      if day > 28 or day < 1:
        return False
  return True
```

```
def verify_id(ids):
  if verify_char_length(ids):
    if all([verify_last_num(ids),verify_area(ids),verify_date(id
    ↪  s)]):
      print("VALID")
      return True
    else:
      print("INVALID")
      return False
  else:
    print("INVALID")
    return False

# https://www.dute.org/fake-id-card-number
verify_id(ids = "11011319870212l018") # VALID
verify_id(ids = "11011319870312l018") # INVALID
```

递归函数: 在函数体内直接或间接调用自身的函数。

递归思想: 将一个大型的、复杂的问题层层转换为一个与原问题相似的小规模问题来求解，给出一个自然、直观、简单的解决方案。



递归函数的特点

- 使用**选择结构**将问题分成**基础情况和可继续分解情况**；
- 有**一个或多个基础情况**用来结束递归；
- 可继续分解情况通过调用函数自身 (递归) 被分解为**规模更小、与原问题性质相同**的子问题；
- 每次递归调用会不断接近基础情况，直到**变成基础情况，终止递归**。

```python
# 求阶乘
def fact(n):
  # if双选结构
  if n == 0: # 只有一种基础情况
    return 1
  else:
    return n * fact(n-1) # 非基础情况递归调用自身
```

斐波那契数列由 0 和 1 开始，之后的斐波那契数就是由之前的两数相加而得出，前几个斐波那契数是：0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377 ,610, 987。

```
def fibo(n):
  # 多选结构
  if n == 0: # 基础情况1
    return 0
  elif n == 1: # 基础情况2
    return 1
  else:
    return fibo(n-1) + fibo(n-2) # 非基础情况递归调用自身
```

注意：① 编写递归函数时，需要仔细考虑基础情况；② 一般的递归函数会占用大量的程序栈，尤其是当递归深度特别大的时候，有可能导致栈溢出；③ 当递归不能使全部的问题简化收敛到边界情况时，程序就会无限运行下去并且在程序栈溢出时导致运行时错误 (递归函数写错了)；④ 掌握递归必须多练习，常应用。

课堂练习

1. 请定义函数，使用循环结构计算 n 的阶乘。

2. 请定义函数，使用循环结构计算第 n 个斐波那契数。

3. 请定义函数，使用循环结构输出包含前 n 个斐波那契数的列表。

4. 在梵文中，短音节 S 占一个长度单位，长音节 L 占两个长度单位，请定义函数，找出所有可能的长短音节组合方式，使得组合之后的结果长度为 n。例如 $V_4 = \{LL, SSL, SLS, LSS, SSSS\}$，$V_4$ 可进一步划分为两个子集合：$\{LL, LSS\}$ (将 L 与 $V_2 = \{L, SS\}$ 中的每个元素组合可得)，$\{SSL, SLS, SSSS\}$ (将 S 与 $V_3 = \{SL, LS, SSS\}$ 中的每个元素组合可得)。

课堂练习答案

1. 请定义函数，使用循环结构计算 n 的阶乘。

```
def fact(n):
  result = 1
  for i in range(1,n+1):
    result *= i
  return result
```

课堂练习答案

2. 请定义函数，使用循环结构计算第 n 个斐波那契数。

```
def fibo_loop(n):
  if n == 0:
    return 0
  elif n == 1:
    return 1
  else:
    a = 0
    b = 1
    for i in range(n-1):
      c = a + b
      a, b = b, c
    return c
```

课堂练习答案

3. 请定义函数，使用循环结构输出包含前 n 个斐波那契数的列表。

```
def fibo_list_loop(n):
  if n == 0:
    return [0]
  elif n == 1:
    return [0, 1]
  else:
    a = 0
    b = 1
    l = [0, 1]
    for i in range(n-1):
      c = a + b
      a, b = b, c
      l.append(c)
    return l
```

课堂练习答案

4. 请定义函数，找出长度为 n 的梵文长短音组合。

```python
def virahanka1(n):
    # 第一种基本情况
    if n == 0:
        return [""]
    # 第二种基本情况
    elif n == 1:
        return ["S"]
    else:
        s = ["S" + prosody for prosody in virahanka1(n-1)]
        l = ["L" + prosody for prosody in virahanka1(n-2)]
    return s + l
```

汉诺塔游戏: 将柱子 A 上的 n 个盘子移动到柱子 C 上，需遵守如下规则:

① 一次只能移动一个圆盘;

② 只能移动最顶部的圆盘;

③ 大圆盘必须在小圆盘下;

请思考所需次数最少的移动步骤。点我在线试玩。

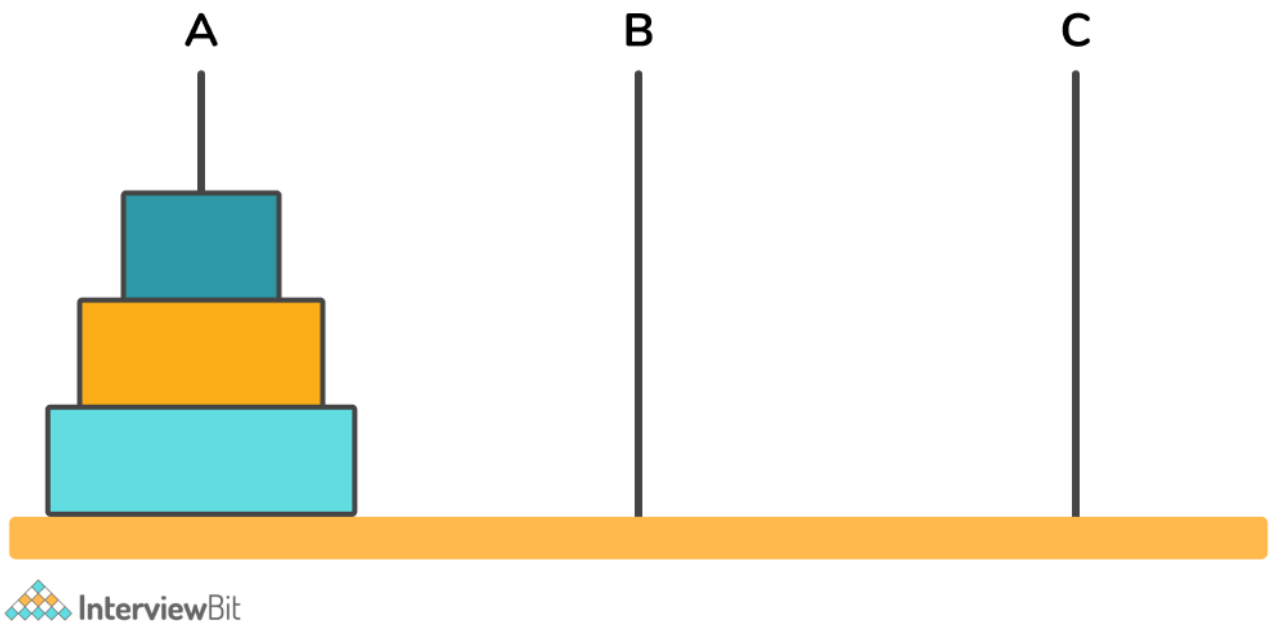


图 1: 3 个盘子的汉诺塔游戏

最小步数的基本思路类似于要把大象装冰箱拢共分三步 ($n \geq 2$)：

① 把前 $n-1$ 个盘子放到 B 柱 (把冰箱门打开)；

② 把第 n 个盘子放到 C 柱 (把大象装冰箱)；

③ 把前 $n-1$ 个盘子从 B 柱放到 C 柱 (把冰箱门带上)。

```python
def move(n, source, target):
  if n == 1:
    print('{}--->{}'.format(source, target)) # 基础情况
  else:
    move(n-1, 'A', 'B') # 把冰箱门打开
    move(1, 'A', 'C') # 把大象装冰箱
    move(n-1, 'B', 'C') # 把冰箱门带上
```

对于 n 个盘子，最少步数为 $2^n - 1 = 2 \times (2^{n-1} - 1) + 1$，$(2^{n-1} - 1)$ 对应上面的第一步和第三步，1 对应第二步。

匿名函数: 使用 `lambda` 表达式创建，语法为

`lambda parameters:expression`

① 参数间用逗号隔开，允许参数有默认值，表达式为匿名函数的返回值，只能由一个表达式组成，不能有其他的复杂结构；

② 可将 lambda 表达式赋值给一个变量，此变量可作为函数使用；

③ 调用 lambda 表达式的语法与调用函数完全相同。

```
(lambda x, y:x**y)(2,3) # 直接调用定义的匿名函数，8
# 使用条件表达式定义匿名函数
my_max = lambda x, y:x if x>y else y
# 定义匿名函数，并将其赋值给变量my_sub
my_sub = lambda x, y=10: x- y
# 调用匿名函数
my_sub(5) # -5
# 使用匿名函数指定sorted函数的排序依据
word_freq = [('my',18), ('go',5), ('can',29), ('exam',3)]
sorted(word_freq,key = lambda item:item[1], reverse=True)
# [('can', 29), ('my', 18), ('go', 5), ('exam', 3)]
```

生成器是创建可迭代对象的一种方式，其保存数据生成的方式 (算法)，有助于节省内存，创建生成器的常用方式有两种：

第一种：将列表推导式的边界符由中括号变为圆括号即可；

第二种：定义生成式函数，生成式函数与一般函数的主要区别在于，返回结果使用 `yield` 子句而非 `return` 语句。

```
import sys
g = (i**2 for i in range(1000))
l = [i**2 for i in range(1000)]
sys.getsizeof(g) # 在内存中占112个字节
sys.getsizeof(l) # 在内存中占8856个字节
```

生成器函数返回一个生成器对象，通常使用 `for` 循环依次获得生成器对象的每一个生成值，也可以通过其他内置函数将其转换为某种可迭代对象 (如，列表，元组)。下面定义一个生成器函数，返回斐波那契数列的前 n 个值。

```
# 先看课堂练习题2的答案
def fibo_loop(n):
  if n == 0:
    return 0
  if n == 1:
    return 1
  else:
    a = 0
    b = 1
    for i in range(n-1):
      c = a + b
      a, b = b, c
    return c
```

将上页代码稍作改动即可得到生成前 n 个斐波那契数的生成器函数。

```
def fibo_generator(n):
  if n == 0:
    yield 0
  if n == 1:
    yield 1
  else:
    a = 0
    b = 1
    yield 0
    yield 1
    for i in range(n-1):
      c = a + b
      a, b = b, c
      yield c

for item in fibo_generator(3):
  print(item)
fibo_list = list(fibo_generator(3))
```

理解生成器的要点

- 使用 `next` 函数查看生成器的下一个生成值，生成器中的代码运行时，每次遇到 `yield` 子句时函数会暂停并保存当前所有的运行信息，返回 `yield` 子句的生成值，并在下一次执行 `next` 函数时从当前位置继续运行；
- 使用 `for` 循环查看生成器的每一个生成值，相当于多次使用 `next` 函数依次查看生成器的生成值，从第一个看到最后一个；
- 当返回最后一个生成值后，再执行 `next` 函数会报错，使用 `for` 循环遍历生成器的每一个生成值时，遍历完最后一个生成值后，结束循环，不会报错，所以，**通常使用 `for` 循环遍历访问生成器的生成值**；(看下一页例子)
- 生成器只能前进，不能回看，就是只能依次生成下一个值。(看下一页例子)

```python
# 定义一个生成器，赋值给变量g，表达式i**2用于计算生成器每次返回的值
g = (i**2 for i in range(2))
next(g) # 0
next(g) # 1
# 根据定义，生成器g只能生成两个值，所以下述代码会报错
next(g) # Error

# 生成器g已经通过上面两次调用next函数，生成了所有值，
↪  此时将其转化为列表，它没有新的元素可生成，所以返回空列表
list(g) # []

g = (i**2 for i in range(3))
# 使用for循环遍历生成器的每一个值
for item in g:
  print(g)

g = (i**2 for i in range(3))
next(g) # 0
list(g) # [1, 4]
list(g) # []
```

```
# 0, 1, 1, 2, 3
# 调用生成器函数，将返回的生成器赋值给变量fibo_g
fibo_g = fibo_generator(5)

next(fibo_g) # 0
next(fibo_g) # 1
list(fibo_g) # [1,2,3]
list(fibo_g) # []
```

`map`、`reduce`、`filter`三个内置函数用于**对序列数据进行批量处理**，优点是可以使代码更简洁，通常用`lambda`表达式 (匿名函数) 来定义如何处理序列中的元素。

注意: 可以使用`for`循环等价地实现这三个内置函数的功能。

`map`使用提供的函数对指定序列做处理，返回对序列中每个元素的处理结果，用法为:

`map(function, iterable, ...)`

```
sent = 'The details of the ICCCT 2023 are as follows'
word_list = sent.split()
m = map(lambda item:len(item), word_list)
type(m) # map
list(m) # [3, 7, 2, 3, 5, 4, 3, 2, 7]
# 计算身份证号校验位的第一步，乘积和
weights = [7, 9, 10, 5, 8, 4, 2, 1, 6, 3, 7, 9, 10, 5, 8, 4, 2]
ids = '11011319870212101 8'
sum(map(lambda x,y:x*int(y),weights,ids)) # 169
```

`reduce` 使用提供的函数 (有两个参数) 对指定序列做处理，返回对序列中全部元素的处理结果，先用前两个元素调用函数，用返回结果与第三个元素继续调用函数，用返回结果与第四个元素继续调用函数，用法为:

`reduce(function, iterable[, init])`

```
from functools import reduce
reduce(lambda x,y:x*y, range(1,6)) # 5! = 120
reduce(lambda x,y:x*y, range(1,6), 2) # 2*5! = 240
reduce(lambda x,y:max(x,y),[8,3,2,9,10,4]) # 10
```

`filter` 使用提供的函数 (返回布尔值 True 或 False) 对指定序列的元素做判断，用序列中的每个元素调用函数，返回结果中只保留能使得函数返回 True 的元素，用法如下:

```
filter(function, iterable)
```

```python
# 只保留由a-z组成的单词，基础版
word_list = ['python', '^_^', 'time.', '3D']
l = filter(lambda item:item.isalpha(), word_list)
type(l) # filter
list(l) # ['python']

# 只保留由a-z组成的单词，完善版
word_list = ['python', '^_^', '我我','time.', '3D']
l = filter(lambda item:all([True if ord('a') <=
↪  ord(char.lower()) <= ord('z') else False for char in item]),
↪  word_list)
list(l) # ['python']
```

THE END