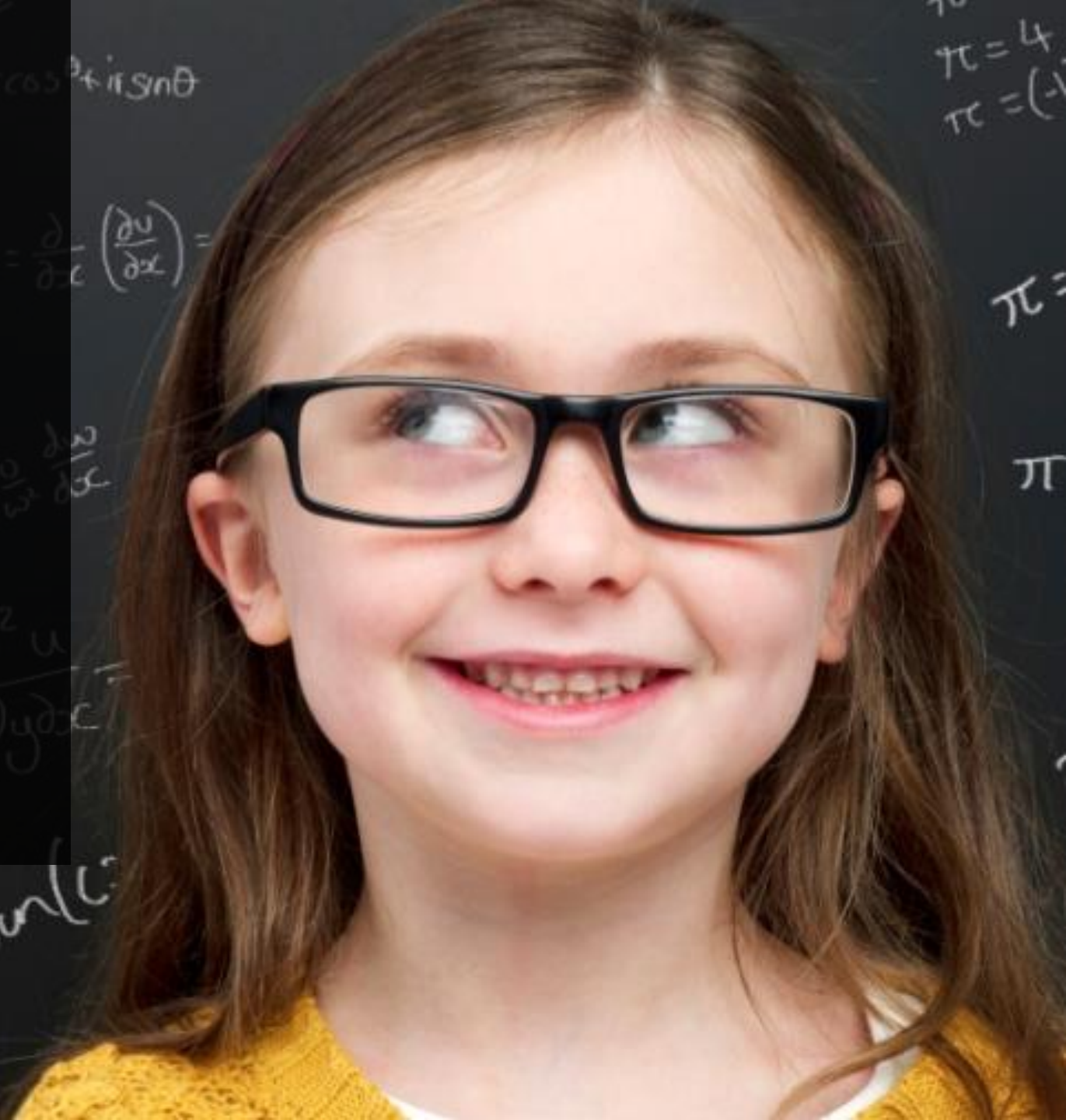


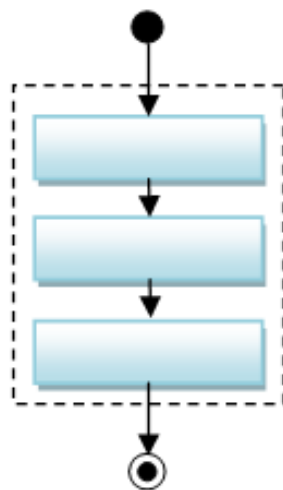
作者：叶蒙蒙



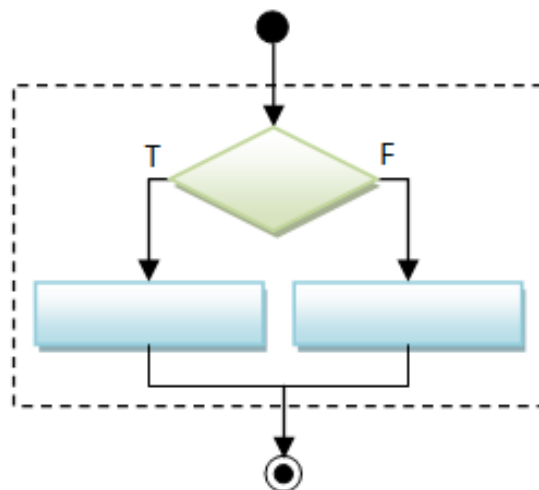


将流程图转化成
代码

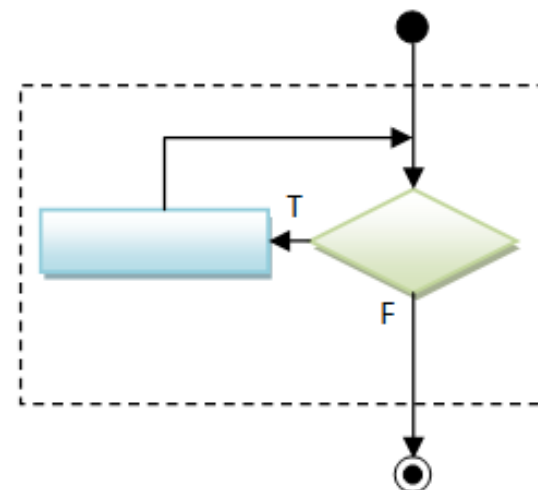
三种控制结构



Sequential

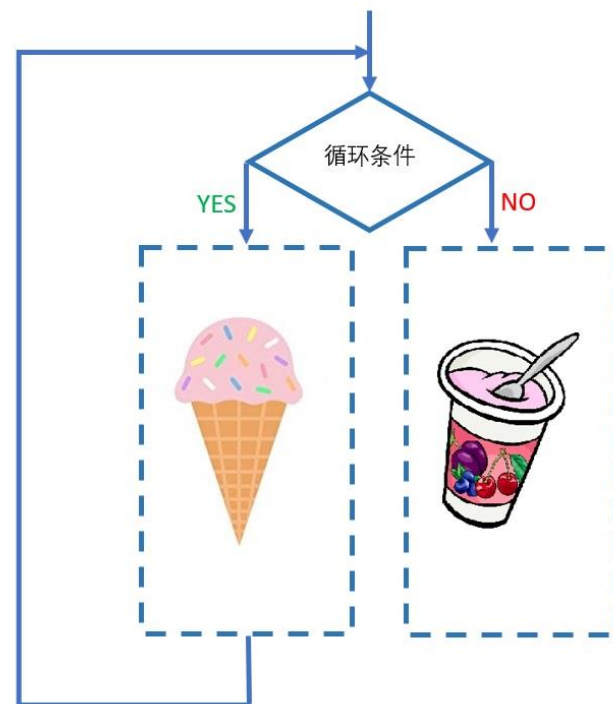
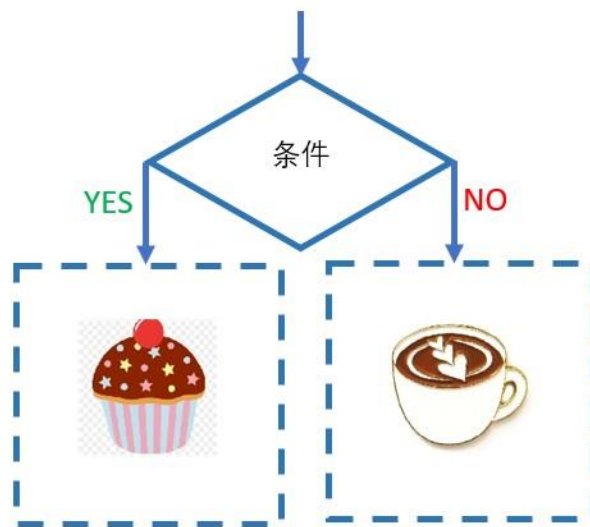
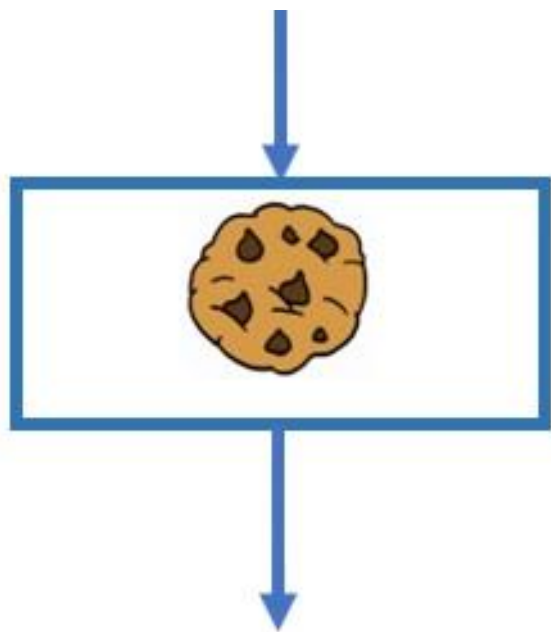


Conditional (Decision)

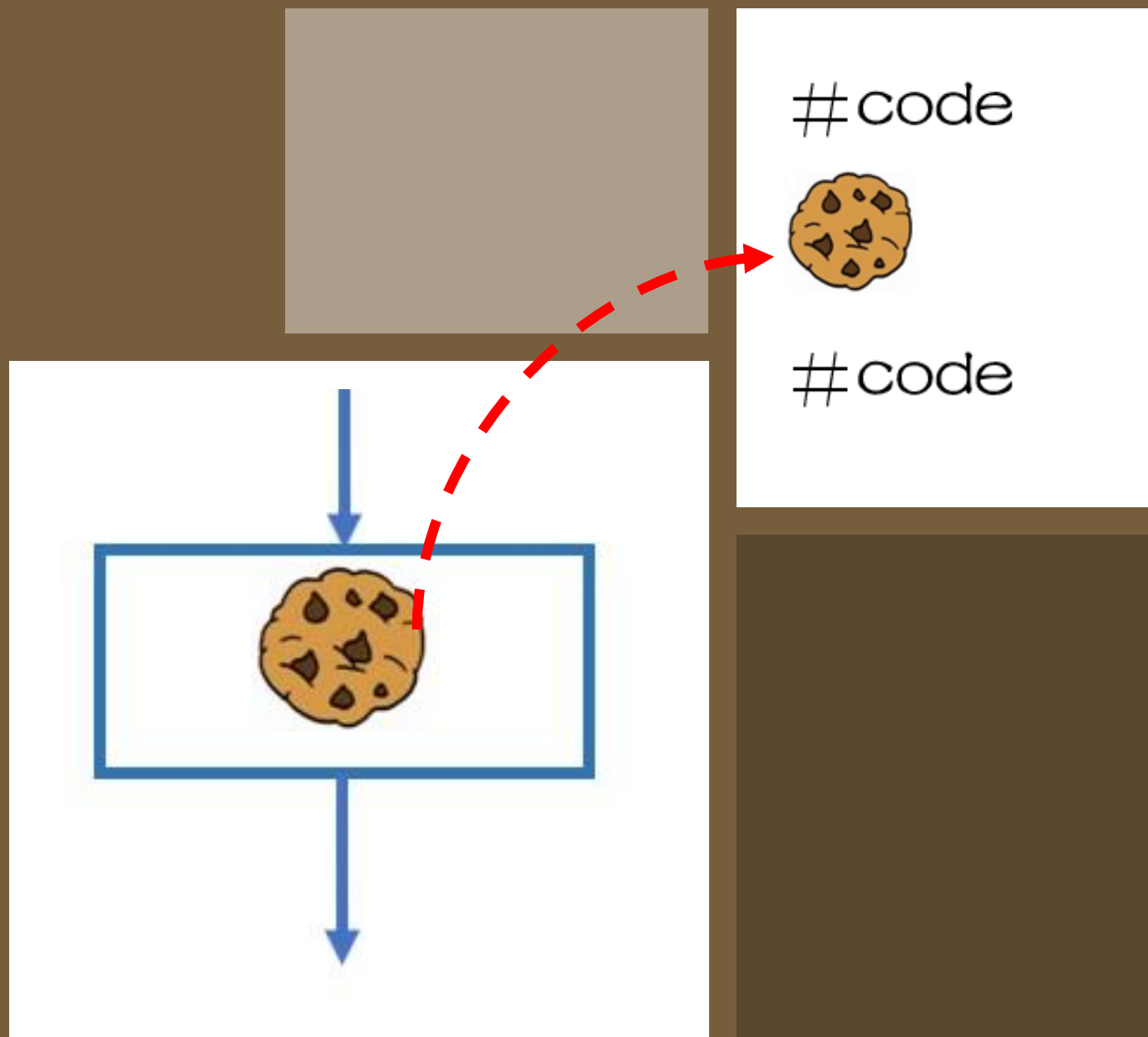


Loop (Iteration)

- 顺序
- 条件
- 循环



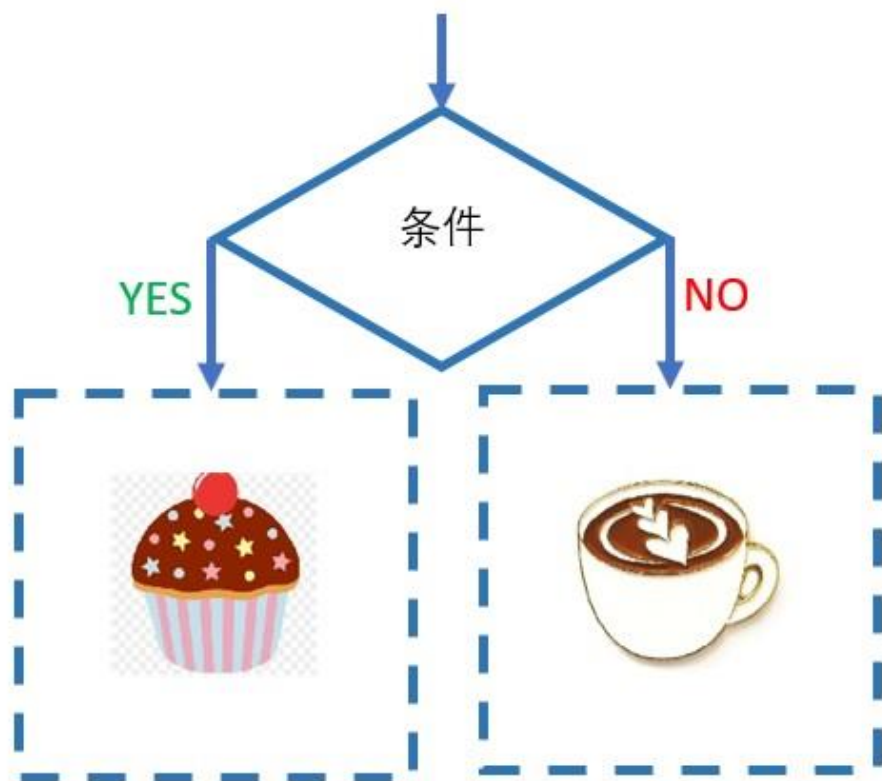
控制流程中的基本结构



顺序结构 到代码

直接抄录

条件结构到代码



```
if ( 条件 ) :
```

后退 4 格

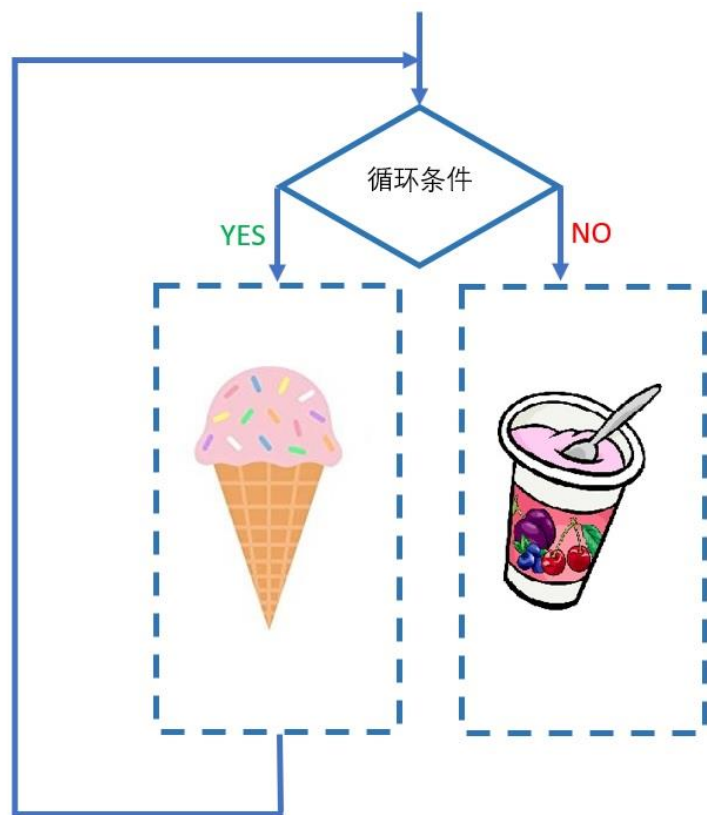


```
else:
```

后退 4 格

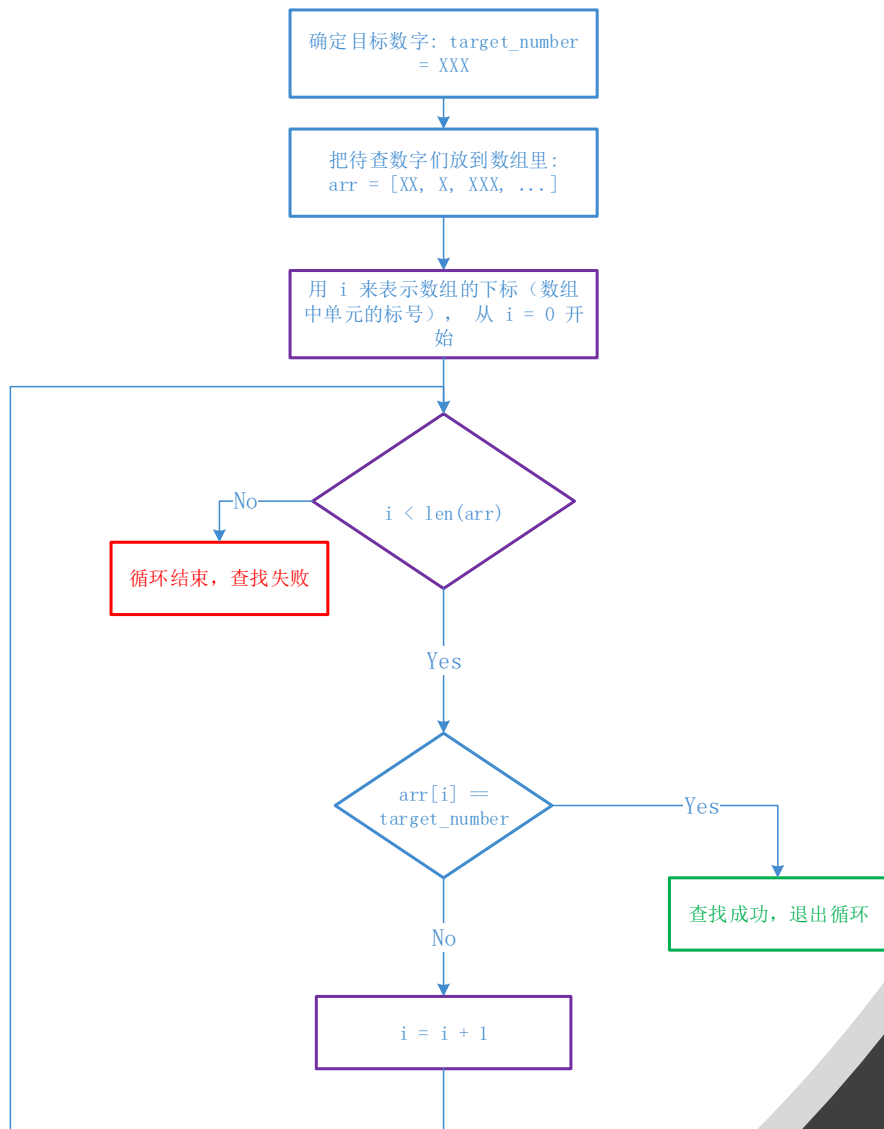


循环结构到代码



while (循环条件):





顺序查找算法从控制流程到代码

```
arr = [38, 17, 26, 4, 2, 18, 66, 73, 84, 45]
target_number = 66
```

```
i = 0
```

```
while (i < len(arr)):
    if (arr[i] == target_number):
        break;
    i = i + 1

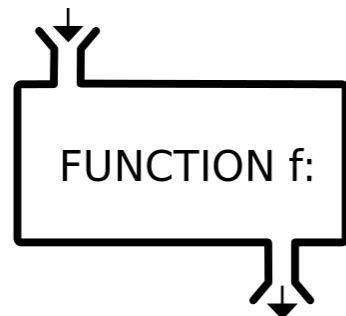
if (i == len(arr)):
    print ("failed")
else:
    print ("succeed to find {0}".format(i))
```


FUNCTIONS



程序中的函数

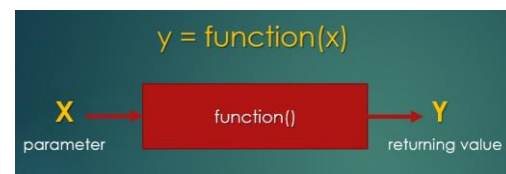
INPUT x

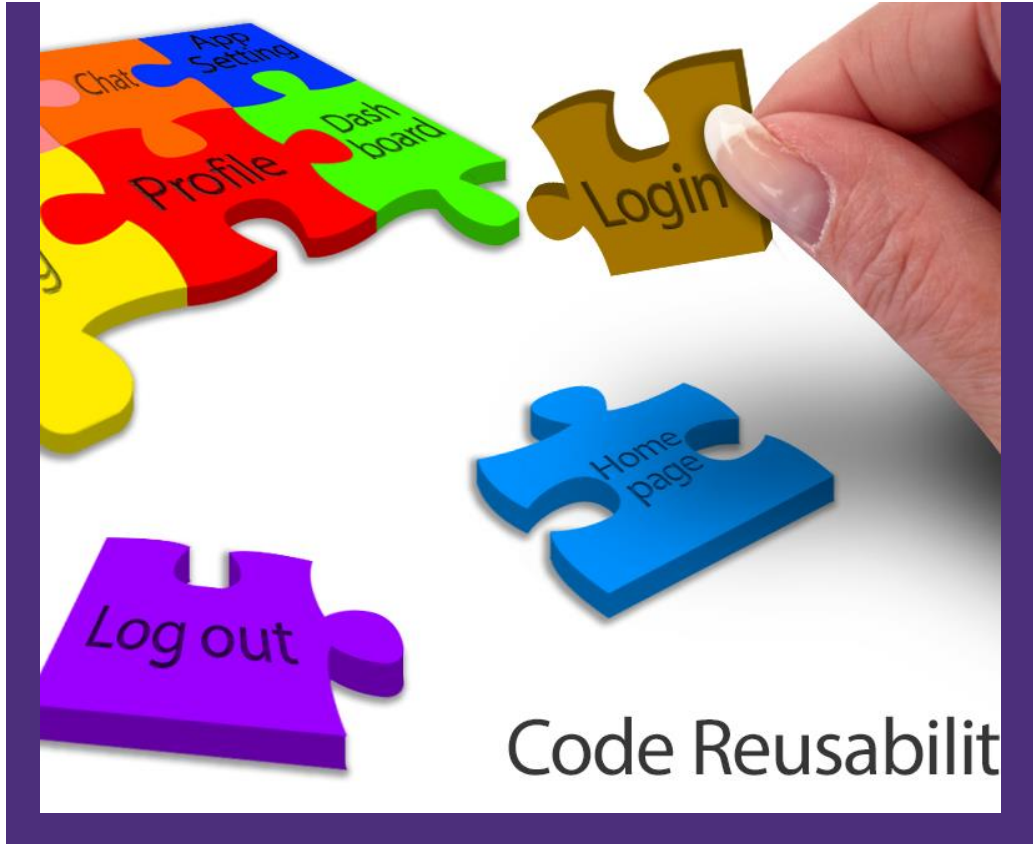


OUTPUT $f(x)$

什么是函数？

- 回忆一下
- 函数
 - 一个命令序列
 - 一个代码块
 - 一个“加工车间”
 - 一个“盒子”
 -





为什么要有函数？

- 复用
 - 同样一块代码（功能），我们要用很多次
- 封装
 - 把一个功能“封起来”，可以当成一件工具，让用的人不用关注里面有什么



def 函数名:

函数体 (代码)

return

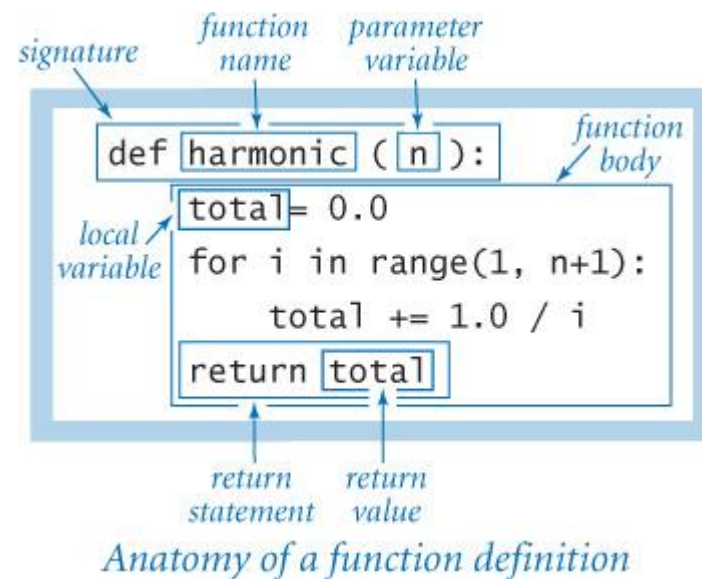
def 函数名 (参数):

函数体 (代码)

return 返回值

组成函数的元素


- 函数包含
 - 必须有的
 - 函数名——函数的名字
 - 函数体——代码块
 - *可以有的
 - 参数——函数的输入
 - 返回值——函数的输出



函数的定义

- 把“盒子”造出来
- 定义函数——把完整的函数“写出来”
 - 函数名
 - 函数体
 - *参数
 - *返回值

Defining Function



Def keyword is required for declaring the functions.

Name of your function. It should be meaningful.

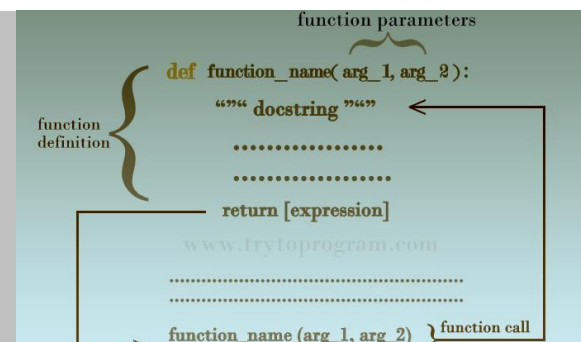
Parameters on which your function will work. It can be empty also.

```
def functionNameHere(parameters):  
    #your code here  
    #your code here  
    #your code here
```

Function scope block statements

函数的调用

- 用“盒子”做事情
- 调用函数
 - 在别的地方“写下”函数的名字
 - 把参数传递给函数
 - 把返回值赋值给一个变量



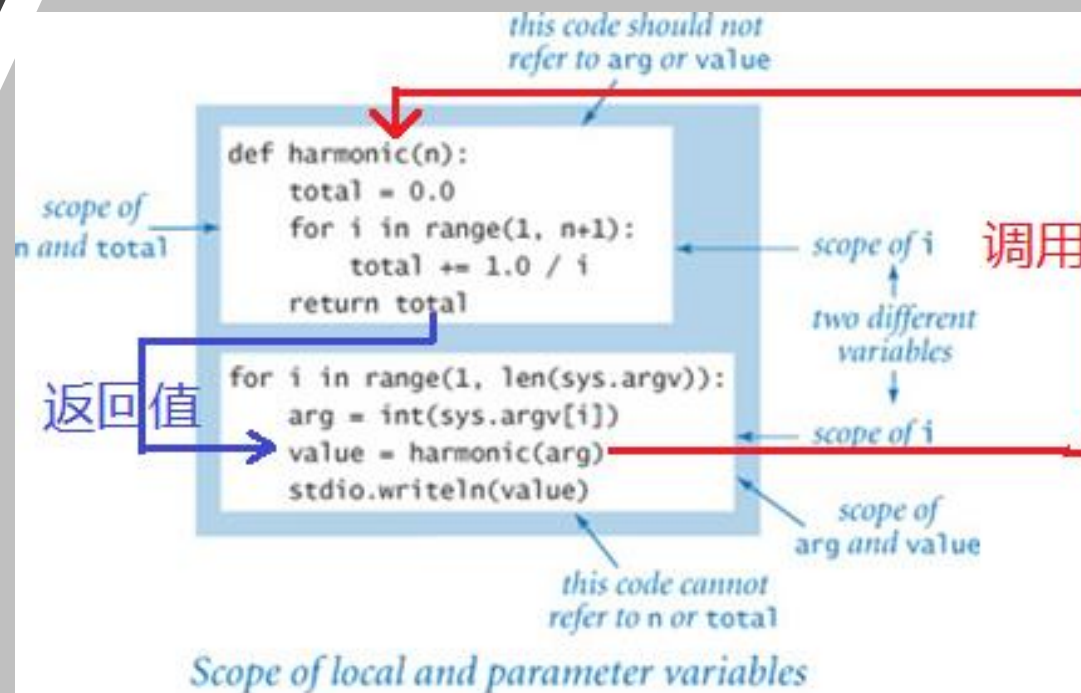
function parameters

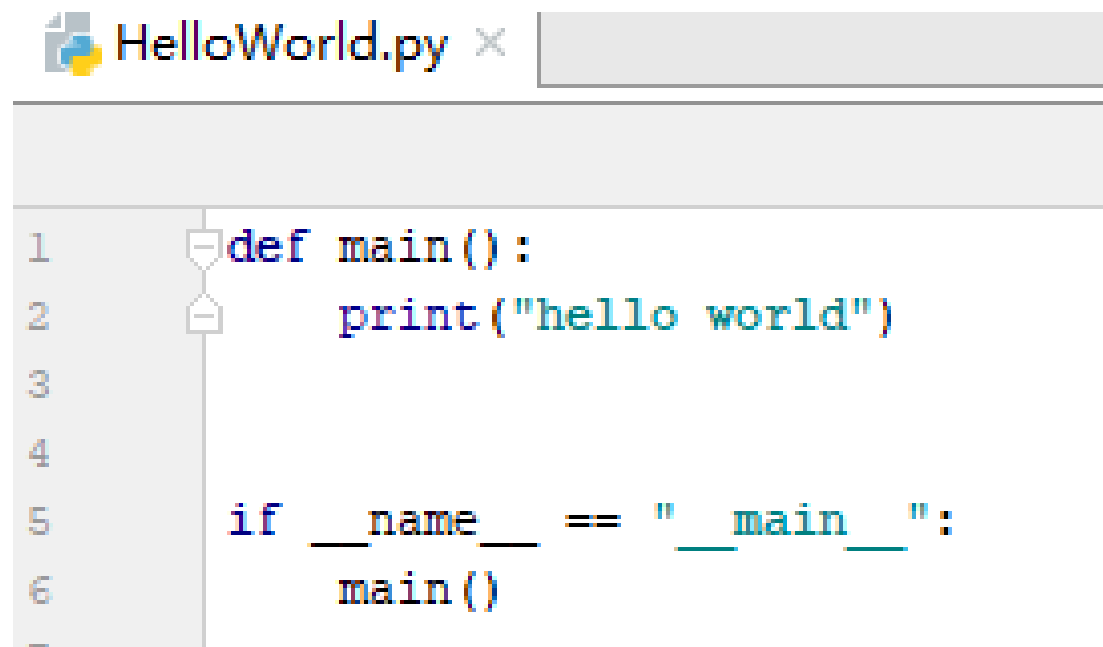
```
def function_name(arg_1, arg_2):  
    """ docstring """  
    .....  
    return [expression]
```

function definition

function_name (arg_1, arg_2) } function call

www.trytoprogram.com





```
1 def main():
2     print("hello world")
3
4
5 if __name__ == "__main__":
6     main()
```

- main() 函数
def main()
- 为什么要写这么麻烦?
 - 为了将来能够运行多个.py文件

Hello World的另一种写法

参数值的传递

pass by reference



fillCup()

pass by value



fillCup()

www.penjee.com

- 把参数传递给函数——在调用函数时，给函数的参数赋值
- 此处传递的参数可以是常量，也可以是变量
- 此处传递的变量可以本身是常量值，也可能本身代表一个数据结构
- (*) 传值和传引用

def 煮(食物):

取一个锅

在锅里注水

把锅放到火上

点火

while (锅里的水没开):

等

把食物放到锅里

while (锅里的食物没熟):

等

熟食物=煮熟的食物

return 熟食物

def 函数名 (参数):

函数体 (代码)

return 返回值

- 函数定义

- 函数名
- 函数体
- 参数
- 返回值

一个例子：“煮”函数·定义

一个例子： “煮”函数·调用

cookedEgg = 煮 (鸡蛋)

food = 汤圆

cookedFood = 煮 (food)

someFoods = [鸡蛋, 西红柿, 土豆, 鱼]

for f in someFoods:

煮 (f)

someFoods

someFoods

| | | | |
|----|-----|----|---|
| 鸡蛋 | 西红柿 | 土豆 | 鱼 |
|----|-----|----|---|

def 混合煮(食物组):

取一个锅

在锅里注水

把锅放到火上

点火

while (锅里的水没开):

等

for f in 食物组:

把 f 放到锅里

index = 0

while (锅里没空):

if (锅里的一种食物熟了):

食物组[index] = 熟了的食物

index = index + 1

等

return

someFoods = [鸡蛋, 西红柿, 土豆, 鱼]

混合煮 (someFoods)



someFoods



另一个例子：“混合煮”
函数

“煮” vs “混合煮”

- 煮 —— someFoods就像一个样品盒，看到其中一种食物，从别处找个同样的来拿去“煮”，循环之后样品盒里的东西都没变

someFoods = [鸡蛋, 西红柿, 土豆, 鱼]

for f in someFoods:

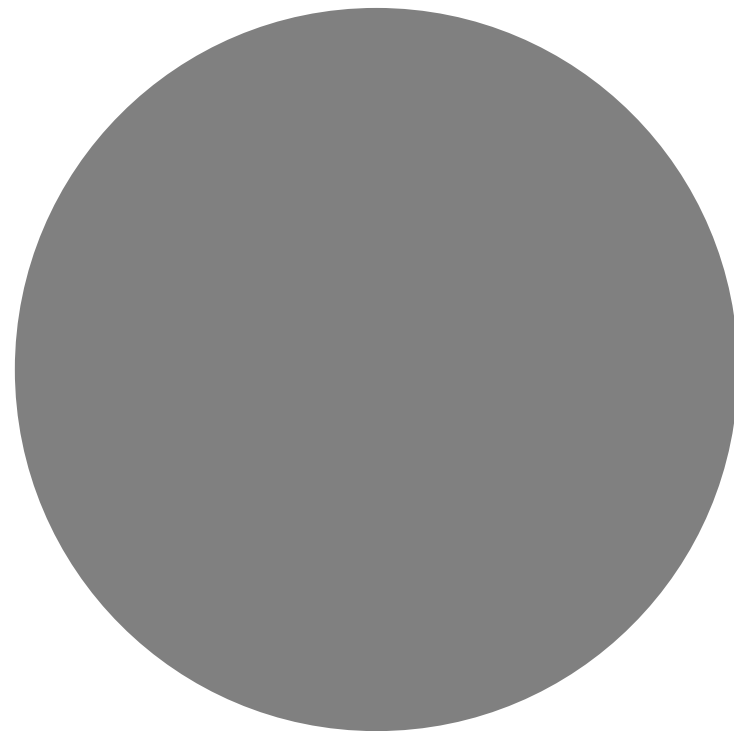
 cf = 煮 (f)

- 混合煮 —— someFoods是实际的食物盒，里面装的东西一起倒到锅里“混合煮”，哪样先熟就把哪样先捞起来放到换来食物盒里，函数运行之后食物盒里面的东西不仅顺序变了，而且都变成“熟的”了

混合煮 (someFoods)

- 习题1：函数：switch(arr, i, j)
 - 功能：交换一个数组中的两个元素。
 - 参数：共三个；arr——数组的名字；i和j——要交换的两个元素的下标
 - 返回值：无请写出用它交换一个名字叫arr_new的数组的第一个和最后一个元素的代码。
- 习题二：函数：findFewest(arr)
 - 功能：找到一个数组中值最小的元素
 - 参数：arr——数组的名字
 - 返回值：最小元素的下标请写出用它搜索名叫arr2的数组中最小值元素的代码

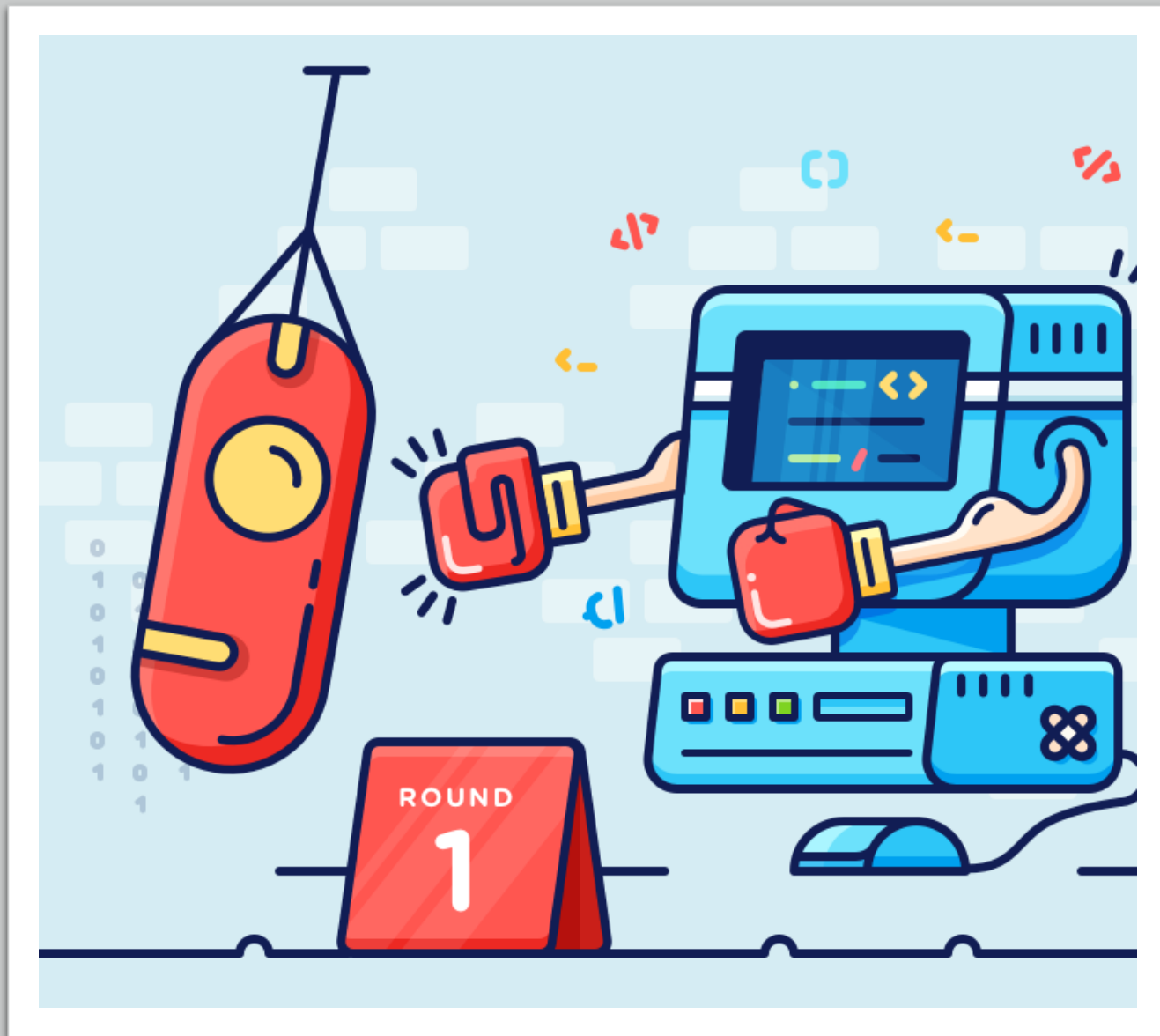
练习题





- 包饺子 的步骤
 - 擀皮
 - 拌馅
 - 包
 - 煮
- Tips
 - “煮”可以调用之前的“煮函数”
 - 其他几步是否也用函数实现？
 - 包饺子函数的参数和返回值分别是什么？
 - 包饺子函数内部调用的其他函数的参数和返回值是什么？

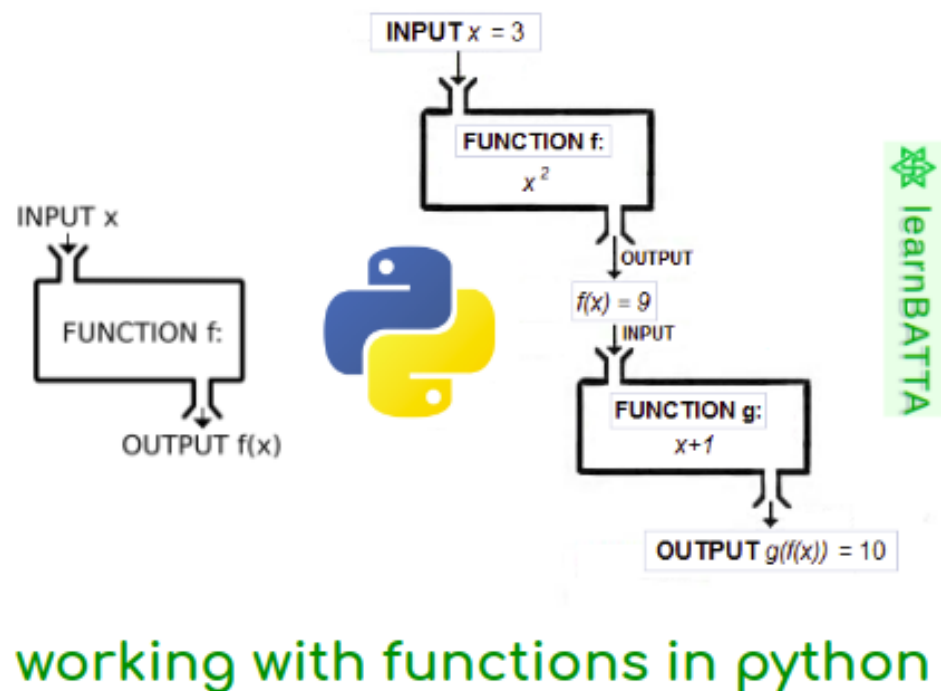
思考题：试着用伪代码写一个包饺子函数



用Python 实现函数

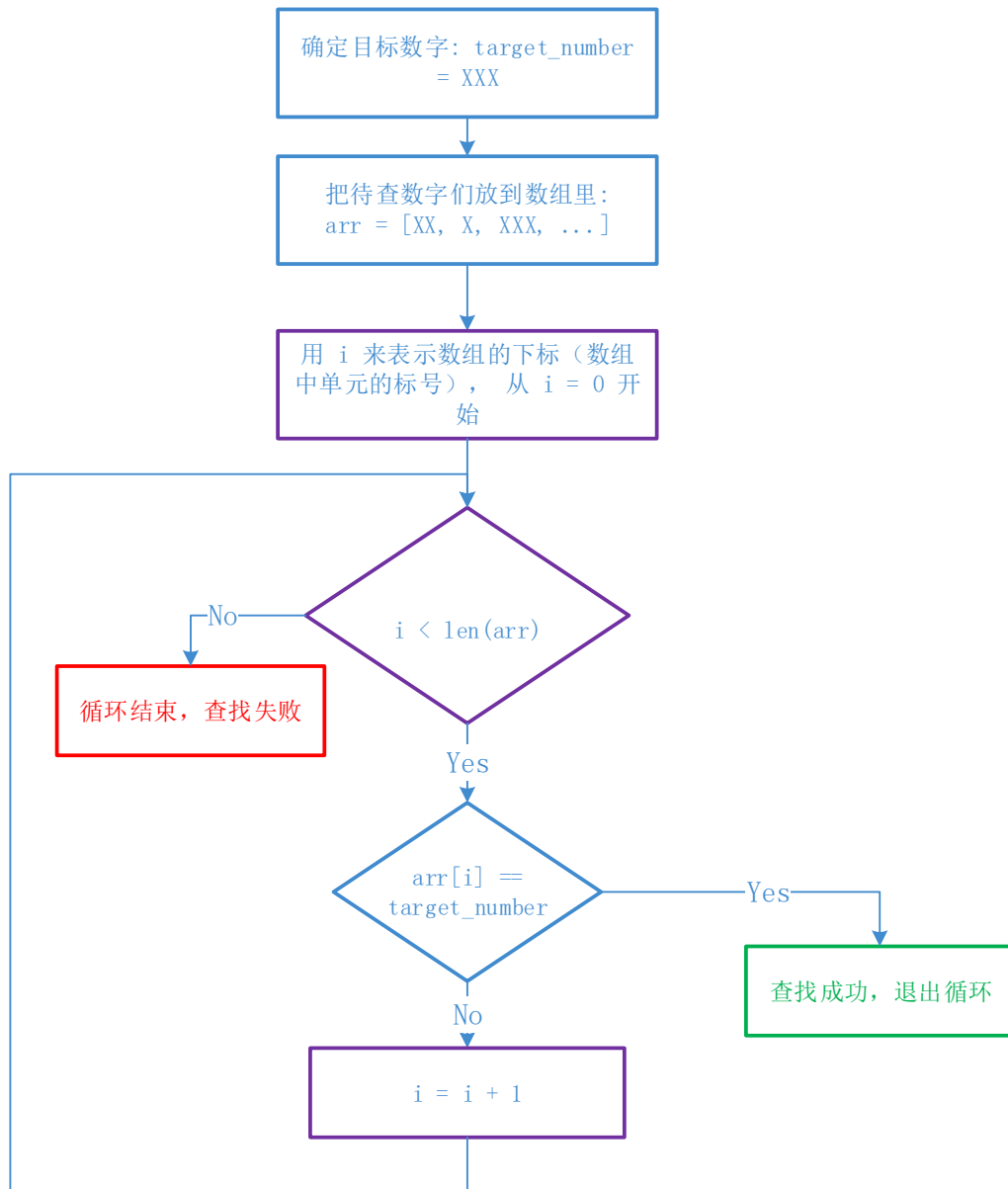
Python中的函数

- 函数
 - 函数名
 - 参数
 - 返回值
 - 函数体
- 函数的参数
 - 参数可以是0个，1个或者很多个
- 函数的返回值
 - 许多编程语言只允许一个返回值
 - Python允许多个返回值



把顺序算法封装成函数

```
def sequential_search(arr, target_number):  
    i = 0  
  
    while (i < len(arr)):  
        if (arr[i] == target_number):  
            break;  
        i = i + 1  
  
    return i
```

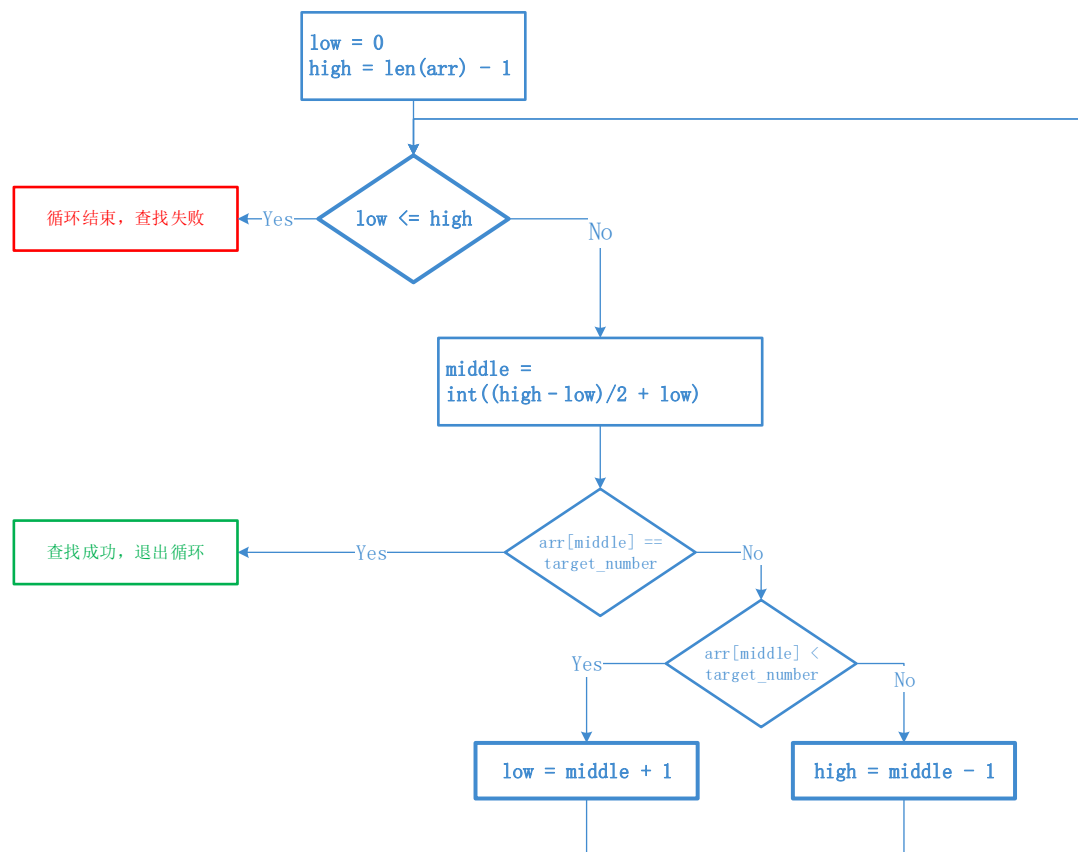


调用顺序查找函数

```
if __name__ == "__main__":  
    arr = [38, 17, 26, 54, 3, 9, 4, 2, 18, 66, 73, 84, 45]  
  
    r = sequential_search(arr, 66)  
    if (r == len(arr)):  
        print ("failed")  
    else:  
        print ("succeed to find {0}".format(r))
```

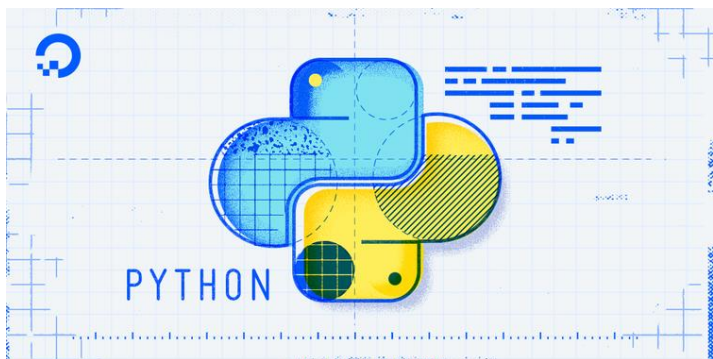


二分查找函数实现



```
def binary_search(arr, target_number):  
    low = 0  
    high = len(arr) - 1  
    while (low <= high):  
        middle = int((high - low) / 2 + low)  
  
        if (arr[middle] == target_number):  
            return middle  
        else:  
            if (arr[middle] < target_number):  
                low = middle + 1  
            else:  
                high = middle - 1  
  
    return -1
```

调用二分查找函数



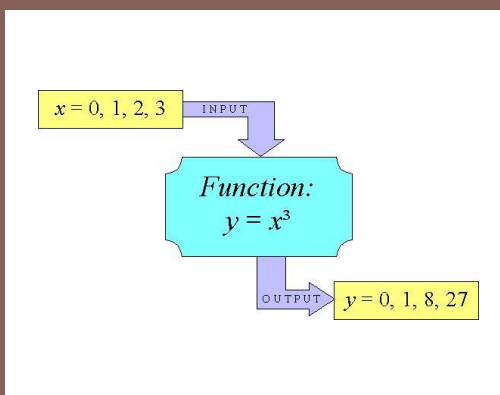
- 将1到1000存储到arr里面去
- 调用binary_search() 函数进行查找

```
if __name__ == "__main__" :  
    arr = []  
    i = 1  
    while (i < 1001):  
        arr.append(i)  
        i = i + 1  
    print(arr)  
    r = binary_search(arr, 6)  
    if (r == -1):  
        print ("failed")  
    else:  
        print ("succeed to find {0}".format(r))
```


Tips: 代码中变量名、函数名使用简称, 易于输入

```
def bs(arr, tn):  
    low = 0  
    high = len(arr) - 1  
    si = -1  
    while low <= high:  
        m = int((high - low)/2 + low)  
        if arr[m] == tn:  
            si = m  
            break  
        elif arr[m] < tn:  
            low = m + 1  
        else:  
            high = m - 1  
    return si
```

```
if __name__ == "__main__":  
    target_number = 0  
    array = []  
    for i in range(1, 1001):  
        array.append(i)  
  
    index = bs(array, target_number)  
  
    if index == -1:  
        print("target number is not found.")  
    else:  
        print("target number is  
{0}.".format(array[index]))
```



谢谢!
