Python程序设计与实践

第三课：循环控制与查找算法

2025. 3

Today

- while循环
- for循环
- 在字符串上进行循环操作
- 猜谜游戏的例子

Recap

- 字符串（STRINGS）
  - 字符串是字符序列，第一个字符的索引是0
  - 字符串可以被索引和分割
- 输入输出操作（INPUT/OUTPUT）
  - 输入通过input命令实现
  - 任何用户输入的内容被Python读入为string对象
  - 输出通过print命令实现，在.py文件只有print出来的才能在终端看到
- 条件分支（CONDITIONS for BRUNCHING）
  - 程序会执行条件返回值为true的代码块
  - 在一个if… elif … elif…的代码结构中，第一个条件为True会被执行
  - 缩进在Python直接影响代码逻辑，一定要注意缩进！

■ 条件分支复习

```
if <condition>:
    < code >
    < code >
    ...
```

```
if <condition>:
    < code >
    < code >
    ...
else:
    < code >
    < code >
    ...
```

```
if <condition>:
    < code >
    < code >
    ...
elif <condition>:
    < code >
    < code >
    ...
elif <condition>:
    < code >
    < code >
    ...
```

```
if <condition>:
    < code >
    < code >
    ...
elif <condition>:
    < code >
    < code >
    ...
else:
    < code >
    < code >
    ...
```

- <condition>只有True或者False两种值
- 执行第一个<condition>为True下面的代码块
- 注意代码缩进

■ 试想一个场景：需要无限地判断当前的状态并做出反馈

- 如果一直往右边走，你将被永远困在森林中
- 如果往其他方向走，你可以走到出口

```
if <exit right>:
    <set background to woods_background>
    if <exit right>:
        <set background to woods_background>
        if <exit right>:
            <set background to woods_background>
            and so on and on and on...
        else:
            <set background to exit_background>
    else:
        <set background to exit_background>
else:
    <set background to exit_background>
```

写不完，根本写不完！

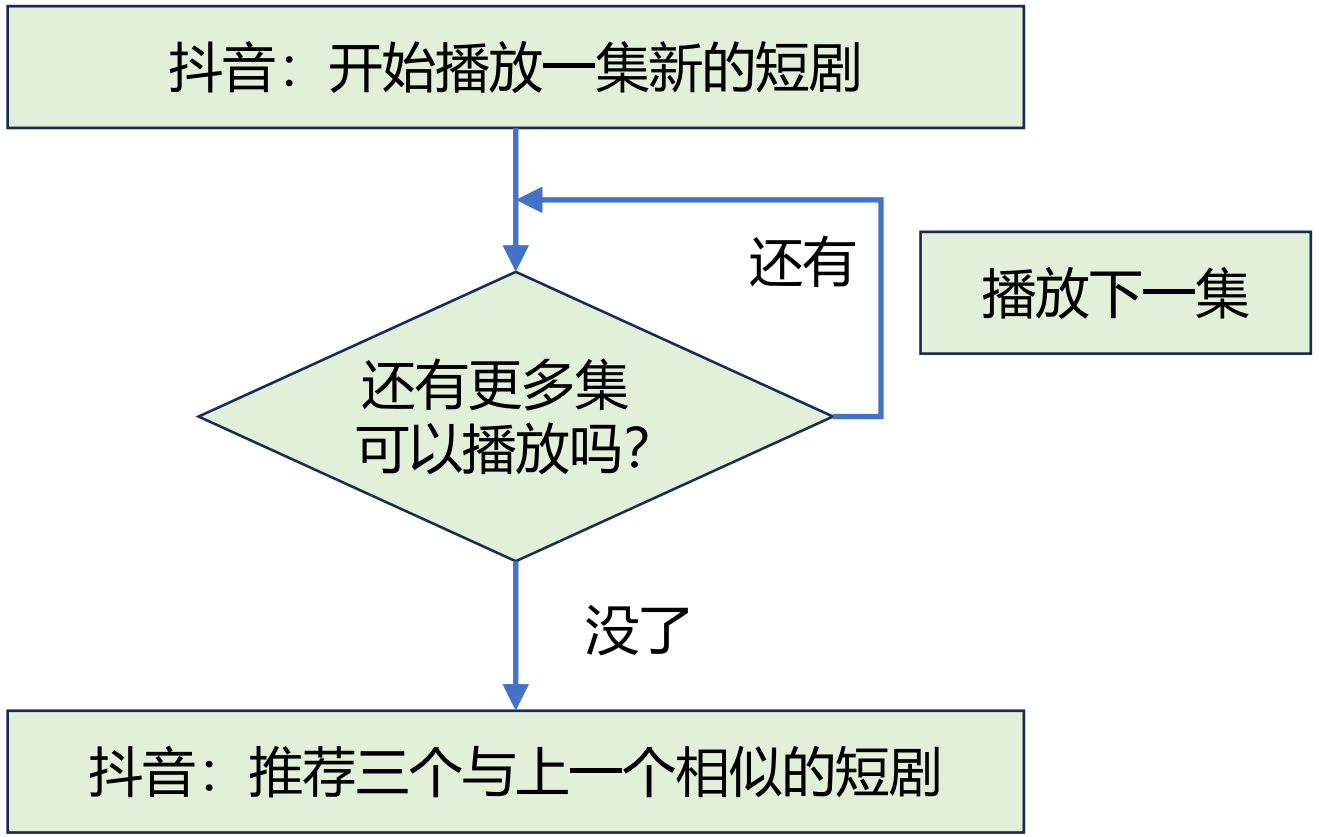
■ 试想一个场景：需要无限地判断当前的状态并做出反馈

- 如果一直往右边走，你将被永远困在森林中
- 如果往其他方向走，你可以走到出口

```
while <exit_right>:
    <set background to woods_background>
    <ask user which way to go>
<set background to exit_background>
```

Chapter 1: `while` Loops

- 场景：沉浸于抖音短剧无法自拔



while loops

■ 流程控制的方法之一：while 循环

```
while <condition>:
    <code>
    <code>
    ...
```

- <condition>的值为布尔类型（True或者False）
- 如果<condition>为True，依次执行while下面所有的代码内容
- 再检查一次<condition>的值
- 重复执行直到<condition>为False
- 如果<condition>永远不为False，将一直循环下去...

while loops

- 流程控制的方法之一：while 循环

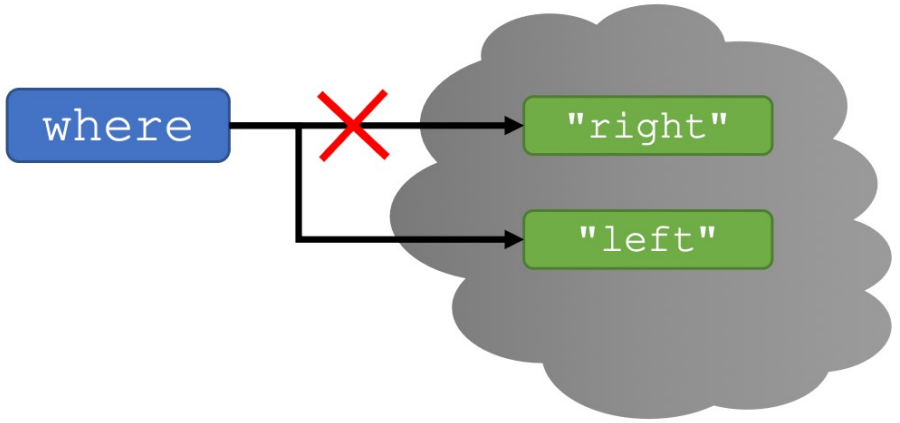
“你被困在了黑暗森林中”

****

**********😂**********

****

“往左走还是往右走？”

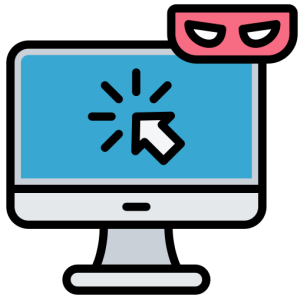



```
where = input(“你被困在了黑暗森林中，往左走还是往右走？”)
while where == “right”:
    where = input(“你被困在了黑暗森林中，往左走还是往右走？”)
Print(“你走出了黑暗森林！”)
```

while loops

- 自己尝试下，如果输入的是“RIGHT”会输出什么?

```
where = input("Go left or right? ")
while where == "right":
    where = input("Go left or right? ")
print("You got out!")
```

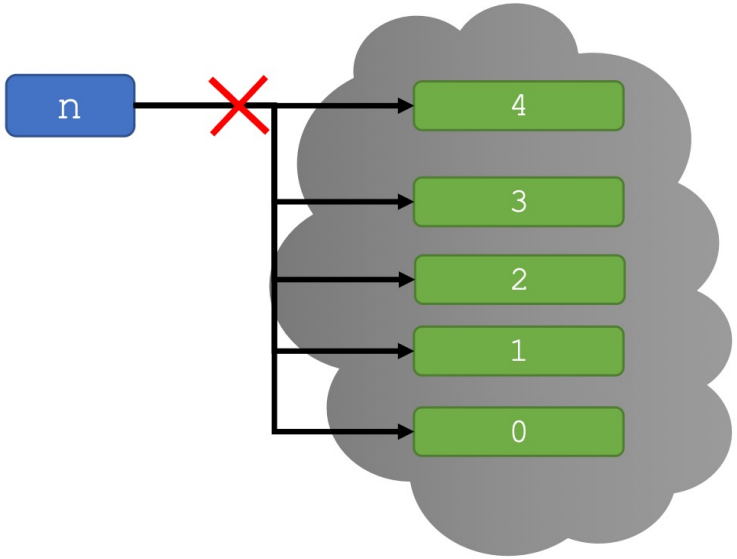


while loops

- while循环的例子

  变量值递减直到满足条件

```
n = int(input("Enter a non-negative integer: "))
while n > 0:
    print('x')
    n = n-1
```



while loops

- while循环的例子

  变量值递减直到满足条件

```
n = int(input("Enter a non-negative integer: "))
while n > 0:
    print('x')
    n = n-1
```

  如果没有最后一行会怎样?

  - 怎么终止循环：按下CTRL-c 或 CMD-c，或者直接叉掉窗口

while loops

- 运行以下代码并终止无限循环

```
while True:
    print("nooooooo")
```

- while循环会无限重复其内部的代码，有时候需要你手动干预

while loops

- 拓展以下代码，当用户执行了两次while循环时，让代码打印出一个sad face ;(
- 提示：定义一个变量作为计数器

```
where = input("Go left or right? ")
while where == "right":
    where = input("Go left or right? ")
print("You got out!")
```

while loops

- 基于while循环流程控制
  - 遍历数字序列执行逻辑

```
n = 0
while n < 5:
    print(n)
    n = n+1
```

在while循环外设置循环变量

检测循环变量是否满足条件

在循环内部对变量进行递增操作

while loops

- 基于while循环流程控制
  - 计算阶乘（factorial）

```
x = 4
i = 1
factorial = 1
while i <= x:
    factorial *= i
    i += 1
print(f'{x} factorial is {factorial}')
```

在while循环外设置循环变量

初始化阶乘值为1

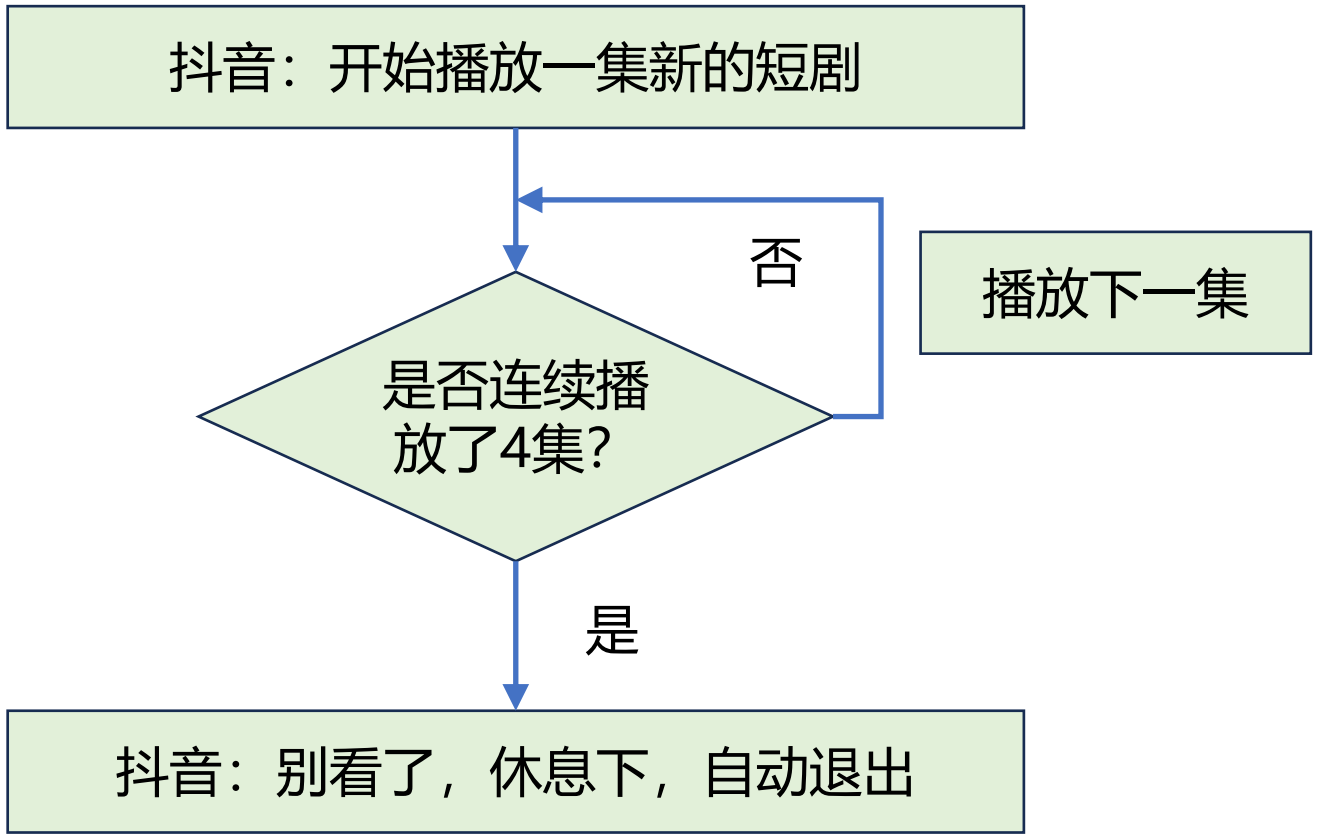
检测循环变量是否满足条件

进行乘法操作

在循环内部对变量进行递增操作

Chapter 2: `for` Loops

- 场景：不让你沉迷于抖音短剧太久



- 流程控制：while 和 for 循环

```
# 用 while 循环略显啰嗦
n = 0
while n < 5:
    print(n)
    n = n+1

# 用for循环比较简洁
for n in range(5):
    print(n)
```

for loops

■ for循环的语法结构

```
for <variable> in <sequence of values>:
    <code>
    ...
```

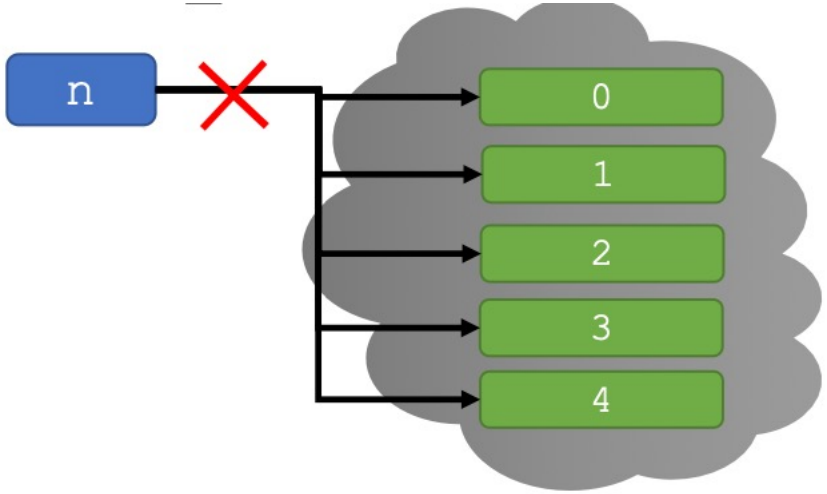
- 每一次循环， <variable> 获取一个值
  - 第一次循环， <variable> 是 <sequence of values>的第一个值
  - 下一次循环， <variable> 是 第二个值
  - 直到<variable> 取完了序列中的所有的值

for loops

- for循环的语法结构

```
for <variable> in range(<some_num>):
    <code>
    <code>
    ...

for n in range(5):
    print(n)
```



- <variable> 从0开始，然后1，2，3，4 (n-1)

- range
  - 根据规则生成一串int类型的数字
  - range(start, stop, step)
    - start：生成的第一个int数字
    - stop：要生成的最后一个数字（不包含这个数字本身）
    - step：隔多长生成下一个数字
  - 跟字符串的分割操作类似，且经常省略start和step
  - 例如：for i in range(4) 从0开始到3结束，step=1
  - 例如：for i in range(3,5) 从3开始到4结束，step=1

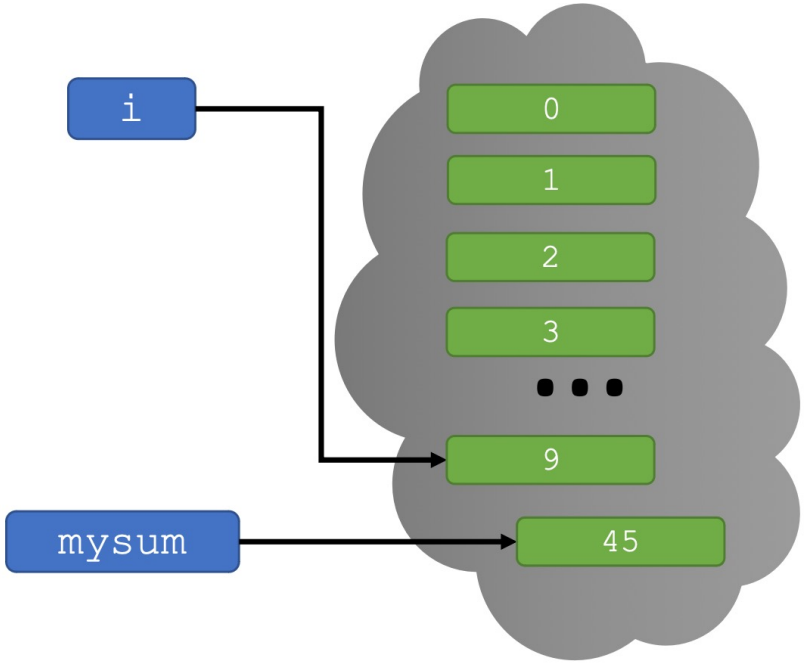
- 请在你的环境中尝试以下语句并查看输出：

```
for i in range(1,4,1):
    print(i)
for j in range(1,4,2):
    print(j*2)
for me in range(4,0,-1):
    print("$"*me)
```

■ 累加操作

- mysum用于存储累加的结果
- range(10)让循环变量从0一步步变为9

```
mysum = 0
for i in range(10):
    mysum += i
print(mysum)
```



- 修改以下代码，让它执行累加操作，从start加到end（包含start和end），如start=3，end=5，那么结果输出12

```
mysum = 0
start = ??
end = ??
for i in range(start, end):
    mysum += i
print(mysum)
```

■ 分别用while和for循环实现阶乘

```
x = 4
i = 1
factorial = 1
while i <= x:
    factorial *= i
    i += 1
print(f'{x} factorial is {factorial}’)
```

之前见过了

```
x = 4
factorial = 1
for i in range(1, x+1, 1):
    factorial *= i
print(f'{x} factorial is {factorial}')
```

for循环只在给定序列的长度内按顺序重复

两种循环的总结：

- While loops：
  - 只要条件为True就一直重复
  - 需要确保不要陷入无限循环
- For loops：
  - 可以在一定范围的数字内重复
  - 还可以在其他一些数据类型上进行循环（字符串、列表等）

■ 想要跳出循环怎么办：

- break语句：

```
while <condition_1>:
    while <condition_2>:
        <expression_a>
        break
        <expression_b>
    <expression_c>
```

- 立刻跳出循环
- 跳过代码块中剩余的语句
- 只退出当前所在的循环代码块

■ 想要跳出循环怎么办：

- break语句：

```
mysum = 0
for i in range(5, 11, 2):
    mysum += i
    if mysum == 5:
        break
        mysum += 1
print(mysum)
```

以上代码break退出到哪里？输出什么？

- **分别用以下range序列编写for循环代码，计算并打印出有几个偶数。**
  - `range(5)`
  - `range(5,6)`
  - `range(2,9,3)`

■ 用代码检查一个字符串中是否有字母 i 或者 u

```
s = "demo loops - fruit loops"
for index in range(len(s)):
    if s[index] == 'i' or s[index] == 'u':
        print("There is an i or u")
```

用range遍历字符串的index

```
for char in s:
    if char == 'i' or char == 'u':
        print("There is an i or u")
```

直接遍历字符串中的字母

```
for char in s:
    if char in 'iu':
        print("There is an i or u")
```

直接遍历字符串中的字母
（更简洁的代码）

- **编写一个程序，让用户输入一个小写的字符串，统计出有几个不一样的字母**
- **例如：用户输入“abca”，程序输出3**

- 提示：
  - 遍历输入字符串的每一个字母
  - 声明一个临时字符串变量
  - 如果当前字母没有在已出现的字母中，将它追加到临时字符串变量上
  - 遍历完后给出临时字符串变量的长度

More Loop Examples

- **展示出用户选择的一个正整数的所有因数**

```
num = int(input("Enter a positive integer: "))
for divisor in range(1, num + 1):
    if num % divisor == 0:
        print(f"{divisor} is a factor of {num}")
```

More Loop Examples

- **运行一个无限循环，直到用户输入了“q”或者“Q”**

```
while True:
    user_input = input('Type "q" or "Q" to quit: ')
    if user_input.upper() == "Q":
        break
```

More Loop Examples

- **展示所有从1到50的数字，除了3的倍数**

```
for i in range(1, 51):
    if i % 3 == 0:
        continue
    print(i)
```

More Loop Examples

- **模拟10000次摇骰子，并展示出要到数字的平均数**

```
from random import randint
num_rolls = 10_000
total = 0
for trial in range(num_rolls):
    total = total + randint(1, 6)
avg_roll = total / num_rolls
print(f"The average result of {num_rolls} rolls is
{avg_roll}")
```

More Loop Examples

- 让用户输入一个整数，如果用户输入的不是整数就重复让用户尝试

```
while True:
    try:
        my_input = input("Type an integer: ")
        print(int(my_input))
        break
    except ValueError:
        print("try again")
```

```
>>> my_input = "abc"
>>> int(my_input)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: invalid literal for int() with base 10: 'abc'
```

More Loop Examples

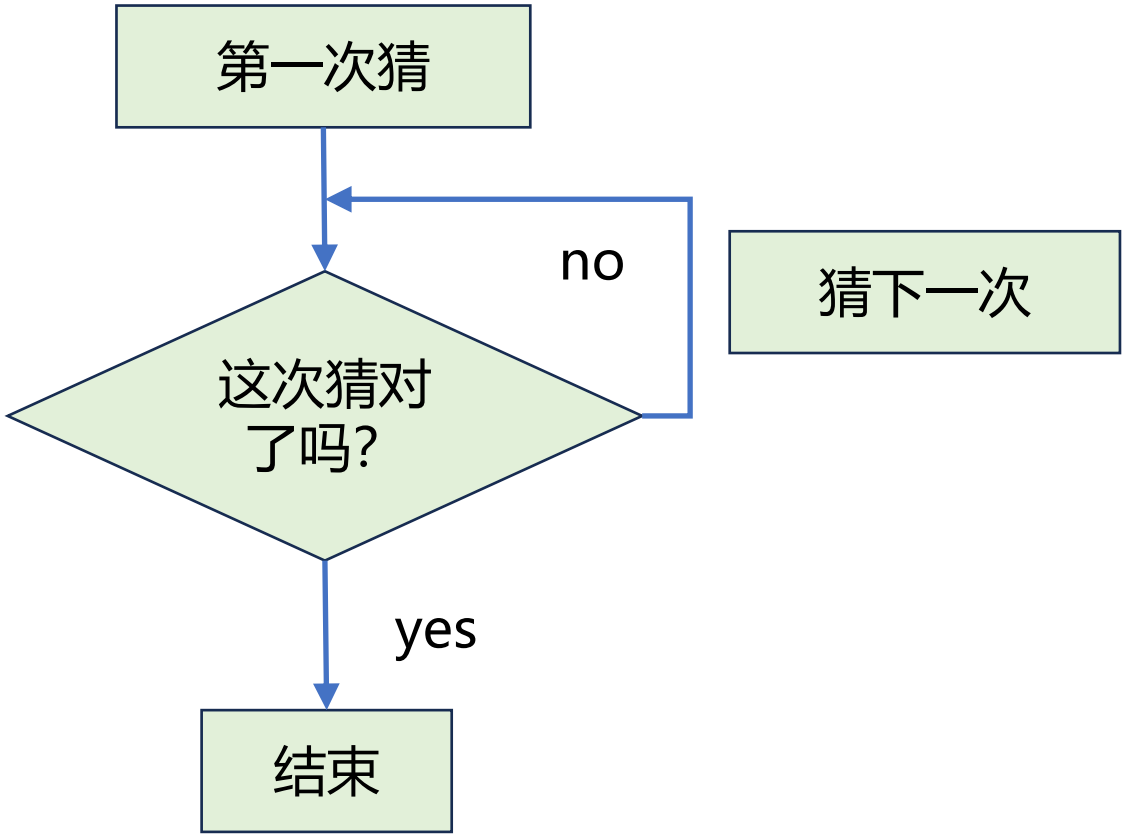
- **让用户输入一个字符串，再让用户输入一个索引值，打印出索引位置的字符**

```
input_string = input("Enter a string: ")
try:
    index = int(input("Enter an integer: "))
    print(input_string[index])
except ValueError:
    print("Invalid number")
except IndexError:
    print("Index is out of bounds")
```

Chapter 3: Guess and Check

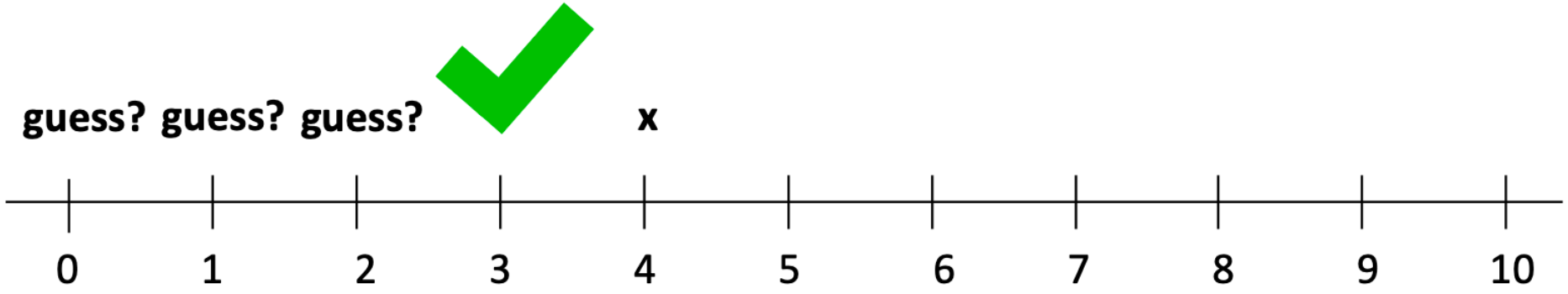
Guess and Check

- 很多场景中需要循环去检查一个逻辑是否正确
- 可以被抽象为 guess and check 的过程

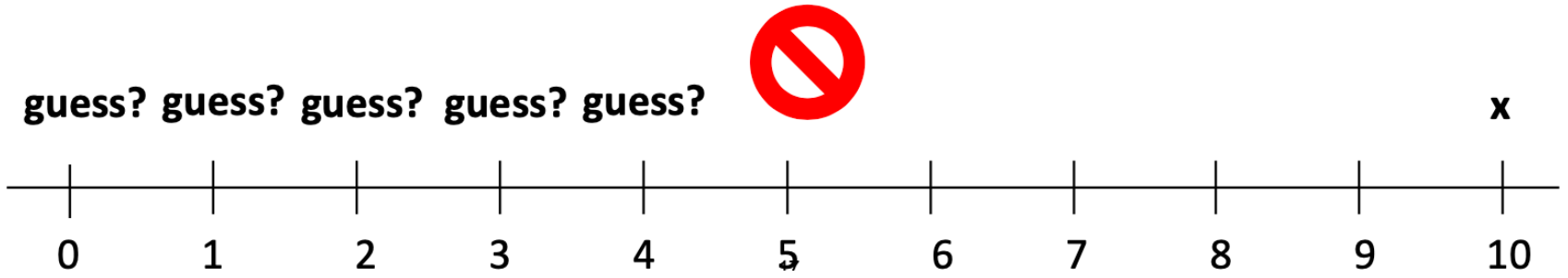


- **Guess and Check**
- **例如：给定一个int整数x，需要找出另一个整数是x的平方根**
  - 先猜一个数字，再检查它是否是答案，依此循环

- **Guess and Check**
- **例如：给定一个int整数x，需要找出另一个整数是x的平方根**
  - 先猜一个数字，再检查它是否是答案，依此循环
  - 我们打算从0开始猜，然后1，2，...
  - 如果x是一个平方数，我们总会找到它的平方根的



- **Guess and Check**
- **例如：给定一个int整数x，需要找出另一个整数是x的平方根**
  - 先猜一个数字，再检查它是否是答案，依此循环
  - 我们打算从0开始猜，然后1，2，...
  - 如果x不是一个平方数怎么办?
  - 需要告诉计算机什么时候停止，比如猜的数的平方大于x就停



- **Guess and Check**
- **例如：给定一个int整数x，需要找出另一个整数是x的平方根**
  - 希望用while循环实现以上逻辑

```
guess = 0
x = int(input("Enter an integer: "))
while guess**2 < x:
    guess = guess + 1
if guess**2 == x:
    print("Square root of", x, "is", guess)
else:
    print(x, "is not a perfect square")
```

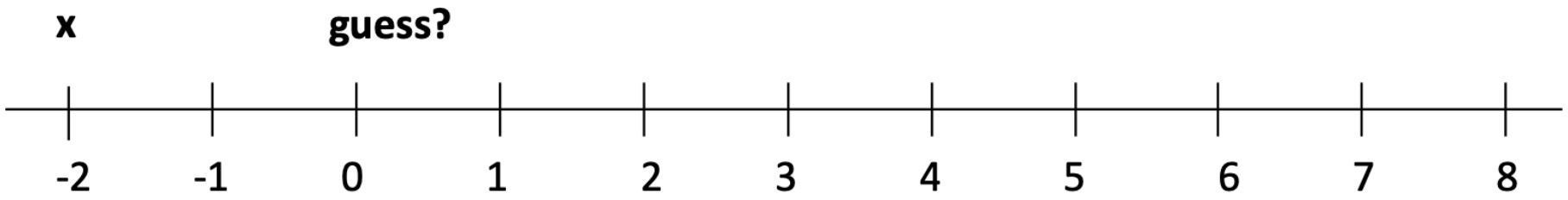
给出了停止猜的条件

输出了停止猜的原因

输出了停止猜的原因

- **Guess and Check**
- **例如：给定一个int整数x，需要找出另一个整数是x的平方根**
  - 这个过程适用于所有整数吗?
  - 如果x是负数怎么办?

    while循环直接退出
  - 希望能处理负数的情况，让程序有不同的表现



- Guess and Check
- 例如：给定一个int整数x，需要找出另一个整数是x的平方根
  - 还是用while循环实现

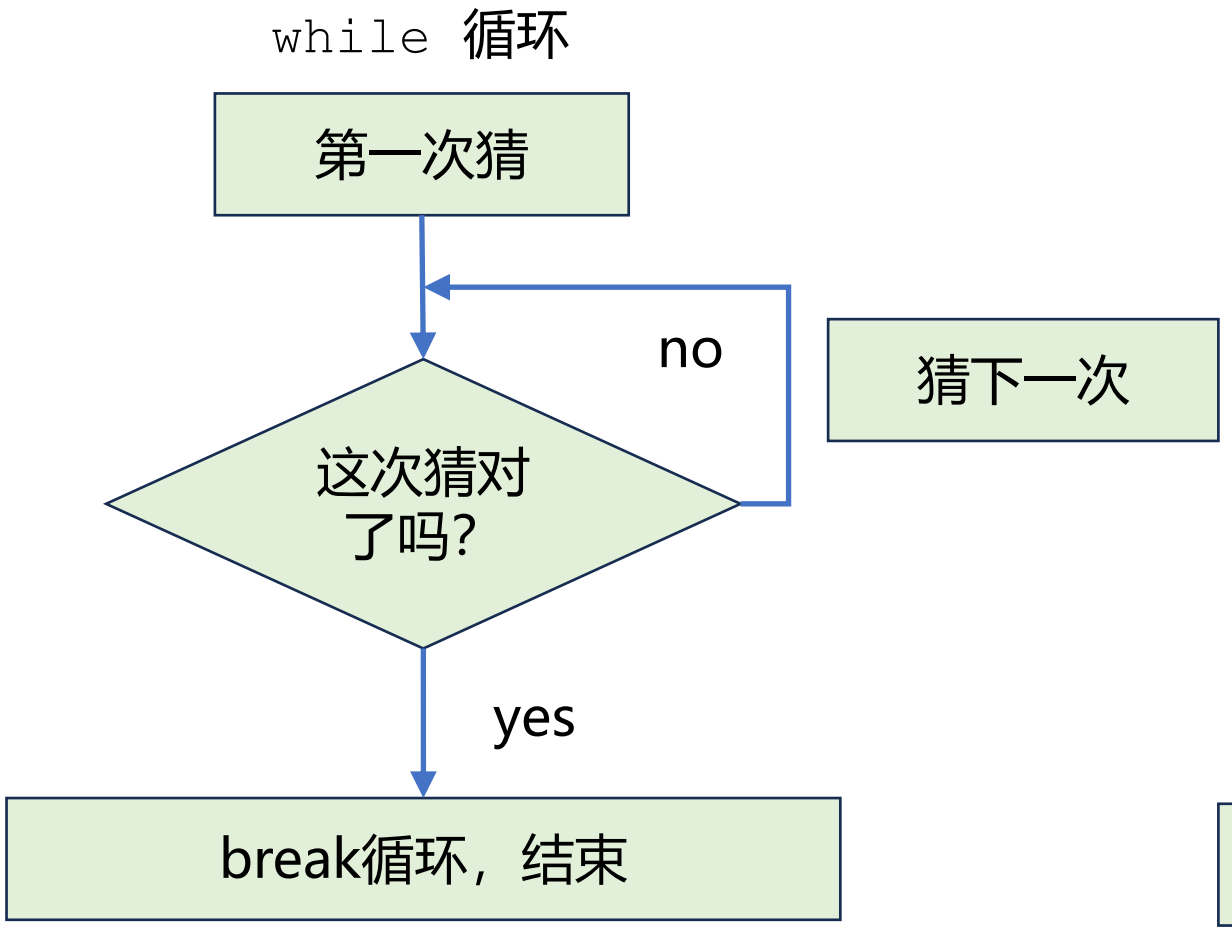
```
guess = 0
neg_flag = False
x = int(input("Enter a positive integer: "))
if x < 0:
    neg_flag = True
while guess**2 < x:
    guess = guess + 1
if guess**2 == x:
    print("Square root of", x, "is", guess)
else:
    print(x, "is not a perfect square")
    if neg_flag:
        print("Just checking... did you mean", -x, "?")
```

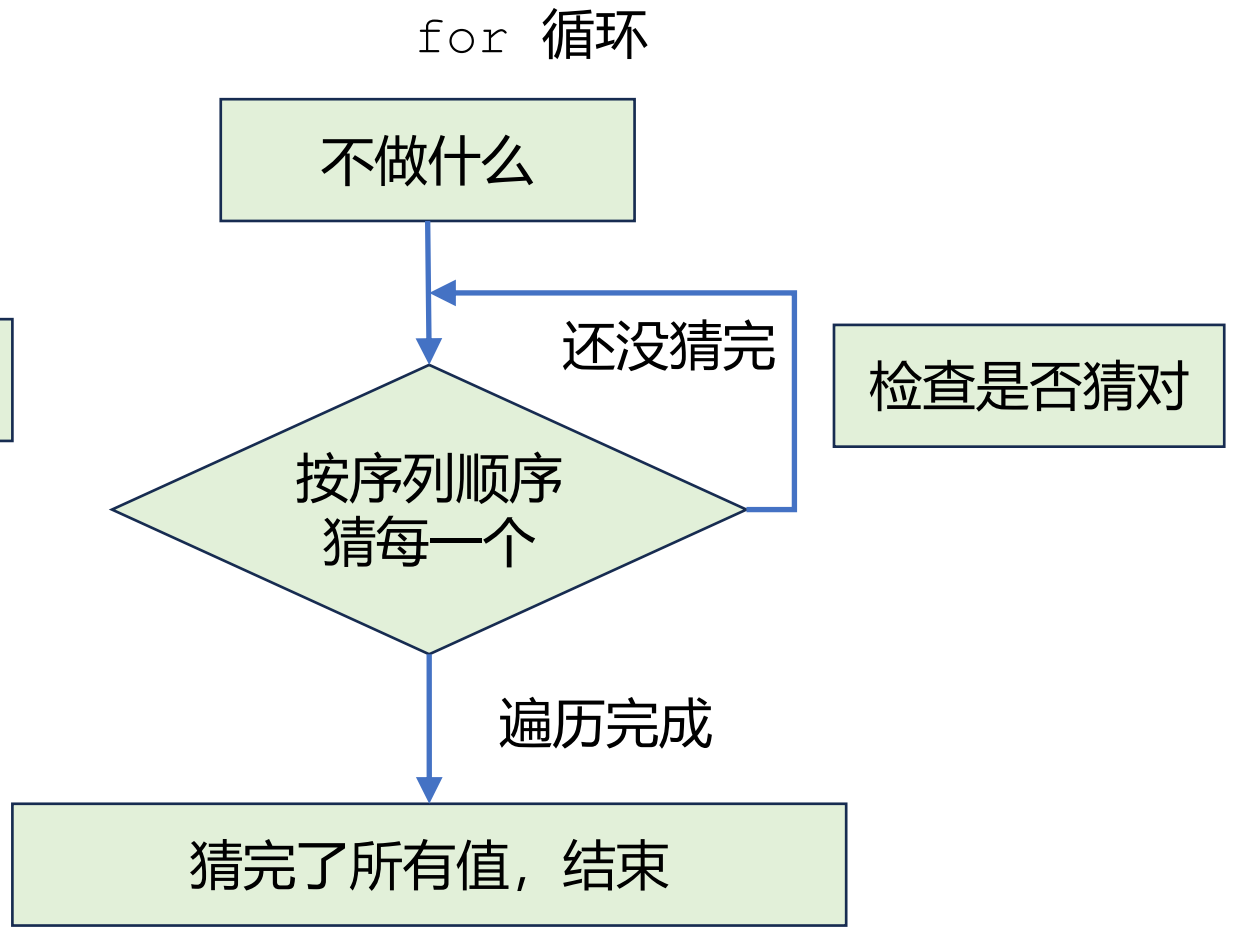
设置变量标记是否为负数

判断负数并修改标记

针对负数给出停止猜的原因

■ Guess and Check





Guess and Check

- 编写一段代码，用户给出一个数字，然后循环从1到10，如果找到了这个数字就输出它，否则就输出没找到的信息
- 你可以选择用while循环还是for循环来实现，也可以都写出来
- 提示：可以设置一个布尔类型的变量来标记是否找到了

Guess and Check

- **Guess and Check**
- **例如：给定一个int整数x，需要找出另一个整数是x的立方根**
- **用for循环怎么写?**

```
x = int(input("Enter an integer: "))
for guess in range(x+1):
    if guess**3 == x:
        print(”Cube root of", x, "is", guess)
```

可以处理x=1的情况，注意range的设置

- Guess and Check
- 例如：给定一个int整数x，需要找出另一个整数是x的**立方根**
- 如何处理x为负数的情况?

```
x = int(input("Enter an integer: "))
for guess in range(abs(x)+1):
    if guess**3 == abs(x):
        if x < 0:
            guess = -guess
        print("Cube root of "+str(x)+" is "+str(guess))
```

使用绝对值函数，假设x为正数

找到了立方根，如果x为负数，再转为负数

- Guess and Check
- 例如：给定一个int整数x，需要找出另一个整数是x的**立方根**
- 好像还可以更快一点？输出信息更丰富一些？

```
x = int(input("Enter an integer: "))
for guess in range(abs(x)+1):
    if guess**3 >= abs(x):
        break
if guess**3 != abs(x):
    print(x, "is not a perfect cube")
else:
    if x < 0:
        guess = -guess
    print("Cube root of "+str(x)+" is "+str(guess))
```

只要超过了x的正数就停止猜

停止猜的原因：不是立方数

处理负数的情况

- **Guess and Check**
- **另一个例子：三元一次方程**

  A、B、C三个人卖票

  B比A卖的少2张

  C卖的是A的2倍

  三个人一共卖出去10张

  问：A卖出去几张?

- **Guess and Check**
- **另一个例子：三元一次方程**

```
for a in range(11):
    for b in range(11):
        for c in range(11):
            total = (a + b + c == 10)
            two_less = (b == a-2)
            twice = (c == 2*a)
            if total and two_less and twice:
                print(f"A sold {a} tickets")
                print(f"B sold {b} tickets")
                print(f"C sold {c} tickets")
```

检查三个人的每一种数量组合

在每一种组合下设定约束条件

如果某一种组合满足约束条件就找到了答案

- **Guess and Check**
- **另一个例子：三元一次方程**
  - 如果总票数很大，组合情况以指数递增（n的三次方），循环很慢
  - 如何减少循环次数呢?
  - 我们发现B和C的卖票数都可以通过A的票数计算得到
  - 只对A进行循环即可

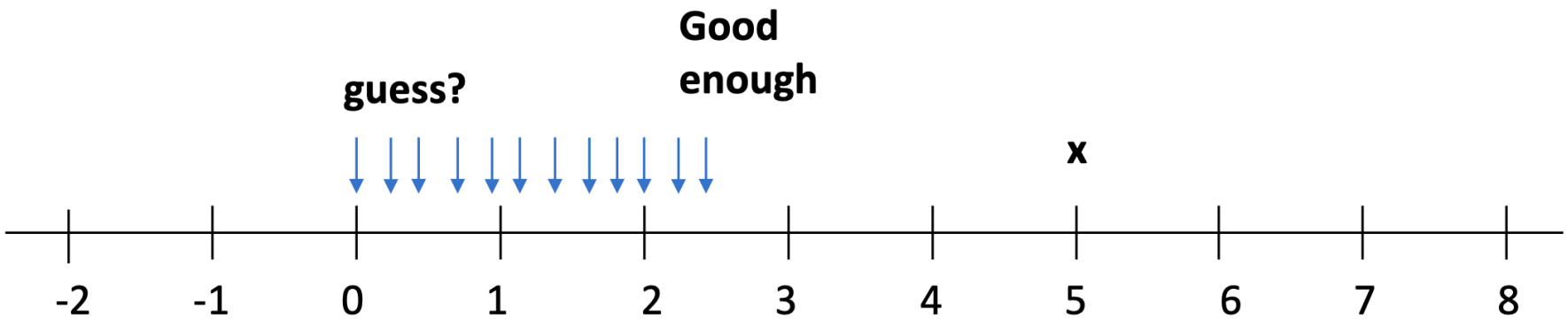
- **Guess and Check**
- **另一个例子：三元一次方程**

```
for a in range(1001):
    b = max(a - 2, 0)
    c = a * 2
    if b + c + a == 1000:
        print("A sold " + str(a) + " tickets")
        print("B sold " + str(b) + " tickets")
        print("C sold " + str(c) + " tickets")
```

Chapter 4: Approximation

Approximation

- **Approximation——趋近**
- 还是之前的猜数字过程
- 精准的答案可能没法得到（谁的平方是10？3还不到，4超过了）
- 需要给出一个方法得到一个足够接近真实答案的结果
- 利用浮点数（float）



Approximation

- **Approximation——趋近**
- 用浮点数来做的话，什么是一个足够好的答案呢?
  - 找到一个数字g，g*g的结果与x足够接近 |g**2 - x| < epsilon
- 利用浮点数找平方根的算法：
  - 先猜一个数字g
  - 每次让g增加一个较小的值increment，重新猜一次
  - 检查是否g**2与x足够接近（|g**2 - x| < e）
  - 循环猜和检查直到足够接近x

Approximation

- **Approximation——趋近**
- 需要设定两个参数：
  - 当前猜的数字举例答案有多近时停止（epsilon）
  - 每次增加多少去接近答案（increment）
- 这两个参数的设置也决定了算法的性能：速度、精度
  - increment越小，程序达到目标的速度越慢，但能得到更精确的结果
  - epsilon越大，得到的结果精度越低，但程序能更快达到目标

Approximation

- **Approximation——趋近**
- 具体实现：

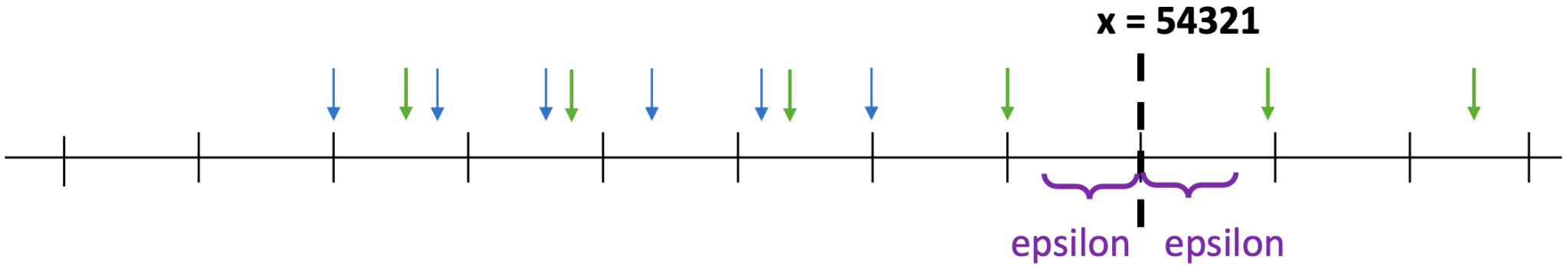
```
x = 36
epsilon = 0.01
num_guesses = 0
guess = 0.0
increment = 0.0001
while abs(guess**2 - x) >= epsilon:
    guess += increment
    num_guesses += 1
print('num_guesses =', num_guesses)
print(guess, 'is close to square root of', x)
```

思考一下：这个循环是否总能退出？

Approximation

- **Approximation——趋近**
- 可以尝试不同的x来查看结果和执行过程的区别
  - 如果x=36，要猜大约60000次
  - 如果x为以下数字，会发生什么?
    - 24
    - 2
    - 12345
    - 54321

- **Approximation——趋近**
- 可以尝试不同的x来查看结果和执行过程的区别
  - 例如x=54321（当x足够大时）
  - 蓝色箭头代表猜的数字
  - 绿色箭头代表猜的数字的平方
  - `abs(guess**2 - x)`永远大于`epsilon` （x越大，当接近x时的跨度越大）



- **Approximation——趋近**
- 怎么调整代码让它按预期退出循环并输出对应的信息?

```
x = 54321
epsilon = 0.01
numGuesses = 0
guess = 0.0
increment = 0.0001
while abs(guess**2 - x) >= epsilon and guess**2 <= x:
    guess += increment
    numGuesses += 1
print('numGuesses =', numGuesses)
if abs(guess**2 - x) >= epsilon:
    print('Failed on square root of', x)
else:
    print(guess, 'is close to square root of', x)
```

当猜的数字的平方超过了目标就退出循环

退出了循环但没找到符合条件的平方根

退出了循环且找到了符合条件的平方根

Approximation

- **Approximation——趋近**
- 现在也有缺点：会因为平方值超过了目标而输出寻找失败
- 还可以降低increment为0.00001，会检查更多的值，但会拖慢程序
- 而且浮点数小数位较多后，浮点数之间的比较可能产生问题

■ Approximation——趋近算法的总结

- 需要注意循环不能跳过退出条件导致无限循环下去
- 需要注意程序运行效率和计算精度之间的权衡
- 需要在设置参数时考虑结果离正确答案多接近，以及每次更新多大幅度
- 有没有更快且更加准确的方法?

■ 二分查找法（Bisection Search）

- 想象一个游戏，有人在一本448页的书里夹了一张百元钞票，如果你能在8次内猜中在哪一页，钞票就归你了，否则就宣布失败。
- 你能成功的概率大概是1/56

- 如果你每次猜完都有人告诉你猜对了、页数太大、页数太小呢?
- 你能成功的概率大概是1/3

- **二分查找法（Bisection Search）**
  - 假设我们知道答案处于某个区间之内
  - 那么我们每次猜这个区间中间那个值
  - 如果没猜中，检查比中间这个值大还是小
  - 更新猜的区间
  - 重复以上步骤

- 之前的猜数字算法：每次把搜索区间从N降为N-1
- 现在的二分查找法：每次把搜索区间从N将为N/2

■ 二分查找法（Bisection Search）

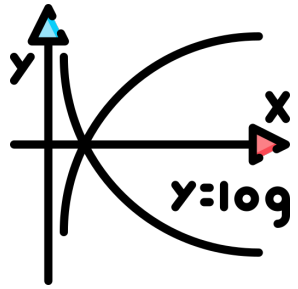
- 之前的猜数字游戏：

  一个挨着一个检查答案，猜的过程是线性变化的（linear）

- 二分查找：

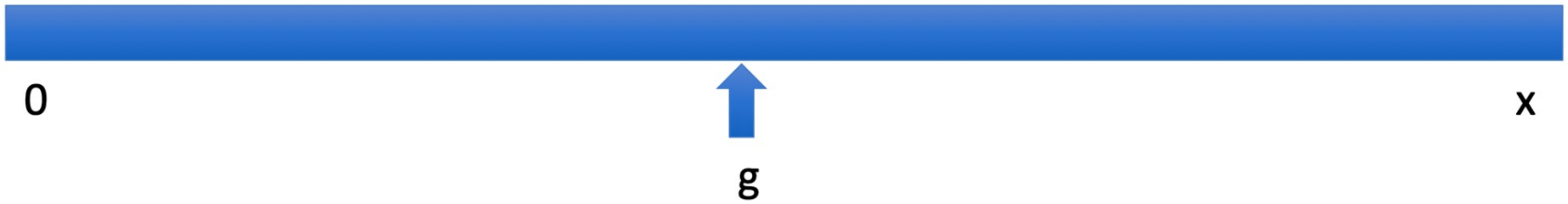
  一半一半地检查答案，猜的过程是对数变化的（logarithmic）

- 对数变化的算法效率更高！



■ 二分查找法（Bisection Search）

- 同样是猜数字游戏
- 假设我们知道答案位于0和x之间
- 我们选取一个处于中间的值作为猜的数字



■ 二分查找法（Bisection Search）

- 如果猜的数字的平方与x足够接近，那我们很幸运直接找到了答案
- 如果猜的数字的平方与x不够接近，**那这个数字是太大了还是太小了?**

  如果g**2 > x，我们知道g太大了，那继续猜一个新的g

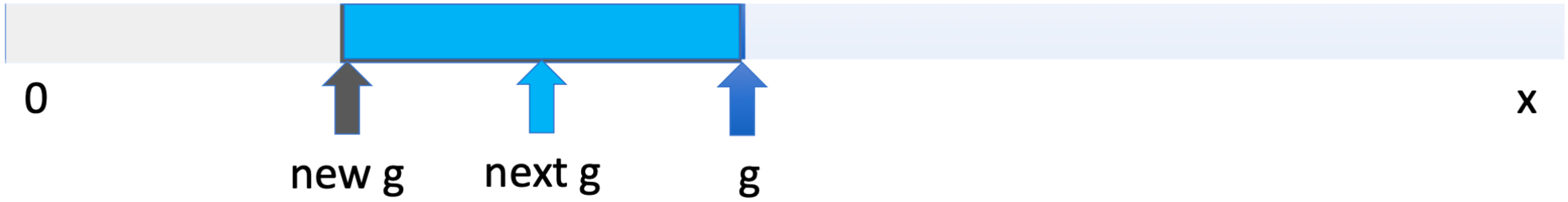
  新的g是0到原来g的一半

■ 二分查找法（Bisection Search）

- 如果猜的数字的平方与x足够接近，那我们很幸运直接找到了答案
- 如果猜的数字的平方与x不够接近，**那这个数字是太大了还是太小了?**

  如果g**2 < x，那我们知道g太小了，继续猜下一个g

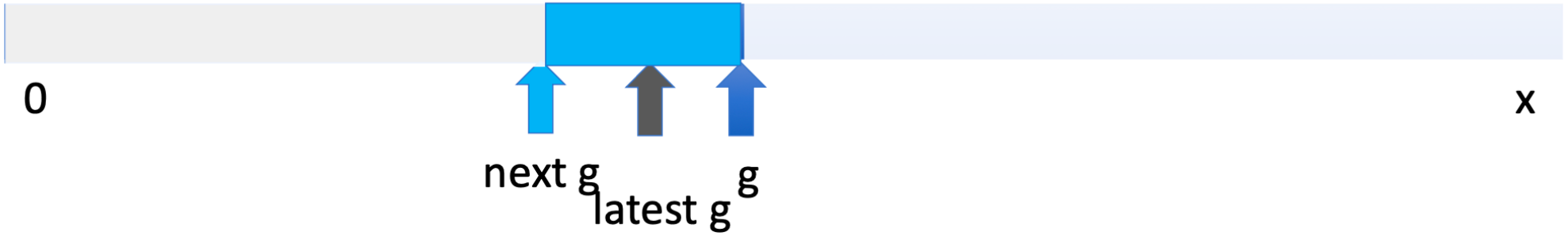
  下一个g是上一个g和第一次g的一半



■ 二分查找法（Bisection Search）

- 如果猜的数字的平方与x足够接近，那我们很幸运直接找到了答案
- 如果猜的数字的平方与x不够接近，**那这个数字是太大了还是太小了?**

  如果g**2 < x，我们知道它还是太小了，继续猜它和第一个g的一半



在每一步，降低搜索的空间为上一步的一半，直到它的平方离x足够近

■ 二分查找法（Bisection Search）

- 二分查找法可行的关键：
  1. 搜索空间是有顺序，可以按顺序查找
  2. 我们可以判断猜的数字太大还是太小

- 尝试给出以下搜索算法的代码：
  - 问题1:猜一个4位数字的密码，唯一的反馈是每次告诉你猜的正确还是错误，你可以用二分查找法找到这个密码吗?
  - 问题2:猜一个0到10之间的任意精度的小数，可以获得的反馈是每次告诉你正确、太大、太小，你可以用二分查找法找到这个小数吗?
  - 请给出以上问题是否可用二分查找解决，如果可以，请尝试给出代码。

■ 二分查找法（Bisection Search）

- 之前猜数字的算法代码

```
x = 54321
epsilon = 0.01
num_guesses = 0
guess = 0.0
increment = 0.00001
while abs(guess**2 - x) >= epsilon and guess**2 <= x:
    guess += increment
    num_guesses += 1
print('num_guesses =', num_guesses)
if abs(guess**2 - x) >= epsilon:
    print('Failed on square root of', x)
else:
    print(guess, 'is close to square root of', x)
```

■ 二分查找法（Bisection Search）

- 我们尝试用二分查找法改写代码

```
x = 54321
epsilon = 0.01
num_guesses = 0
```

定义一些二分查找法需要的变量

```
while abs(guess**2 - x) >= epsilon:
```

每次重复执行一些步骤

```
    num_guesses += 1
print(guess, 'is close to square root of', x)
```

■ 二分查找法（Bisection Search）

- 我们尝试用二分查找法改写代码

```
x = 54321
epsilon = 0.01
num_guesses = 0
low = 0
high = x
guess = (high + low)/2.0
while abs(guess**2 - x) >= epsilon:
    num_guesses += 1
print(guess, 'is close to square root of', x)
```

定义了初始化的最大和最小值，以及第一次猜的数字

■ 二分查找法（Bisection Search）

- 我们尝试用二分查找法改写代码

```
x = 54321
epsilon = 0.01
num_guesses = 0
low = 0
high = x
guess = (high + low)/2.0
while abs(guess**2 - x) >= epsilon:
    if guess**2 < x:

    else:


    num_guesses += 1
print(guess, 'is close to square root of', x)
```

定义了初始化的最大和最小值，以及第一次猜的数字

检查每次猜的数字的平方是太大还是太小

■ 二分查找法（Bisection Search）

- 我们尝试用二分查找法改写代码

```
x = 54321
epsilon = 0.01
num_guesses = 0
low = 0
high = x
guess = (high + low)/2.0
while abs(guess**2 - x) >= epsilon:
    if guess**2 < x:
        low = guess
    else:

    num_guesses += 1
print(guess, 'is close to square root of', x)
```

定义了初始化的最大和最小值，以及第一次猜的数字

如果太小，设置区间的下限为当前猜的数字

■ 二分查找法（Bisection Search）

- 我们尝试用二分查找法改写代码

```
x = 54321
epsilon = 0.01
num_guesses = 0
low = 0
high = x
guess = (high + low)/2.0
while abs(guess**2 - x) >= epsilon:
    if guess**2 < x:
        low = guess
    else:
        high = guess

    num_guesses += 1
print(guess, 'is close to square root of', x)
```

定义了初始化的最大和最小值，以及第一次猜的数字

如果太小，设置区间的下限为当前猜的数字

如果太大，设置区间的上限为当前猜的数字

■ 二分查找法（Bisection Search）

- 我们尝试用二分查找法改写代码

```
x = 54321
epsilon = 0.01
num_guesses = 0
low = 0
high = x
guess = (high + low)/2.0
while abs(guess**2 - x) >= epsilon:
    if guess**2 < x:
        low = guess
    else:
        high = guess
    guess = (high + low)/2.0
    num_guesses += 1
print(guess, 'is close to square root of', x)
```

定义了初始化的最大和最小值，以及第一次猜的数字

如果太小，设置区间的下限为当前猜的数字

如果太大，设置区间的上限为当前猜的数字

在新的区域之间猜一个数字

Approximation

- 如果x是一个0和1之间的小数怎么办？请尝试修改填入以下初始条件

```
x = 0.5
epsilon = 0.01
```

请选择合适的上下限区间

```
guess = (high + low)/2
while abs(guess**2 - x) >= epsilon:
    if guess**2 < x:
        low = guess
    else:
        high = guess
    guess = (high + low)/2.0
print(f'{str(guess)} is close to square root of {str(x)}')
```

总结一下

- 很多场景中，我们需要通过趋近的方式找到一个足够好的答案
  - 浮点数在计算机中进行比较时，不能用==，而需要通过足够接近来比较
- 二分查找法相比逐一遍历更快，但是需要满足一些条件：
  - 需要上下限构成的搜索区间
  - 需要搜索区间是有顺序的
  - 需要每次猜完有反馈（正确、太大、太小）

Reading and QA Time

See you next week !