Python程序设计与实践

第二课：字符串、浮点数、输入输出、条件分支

2025. 2

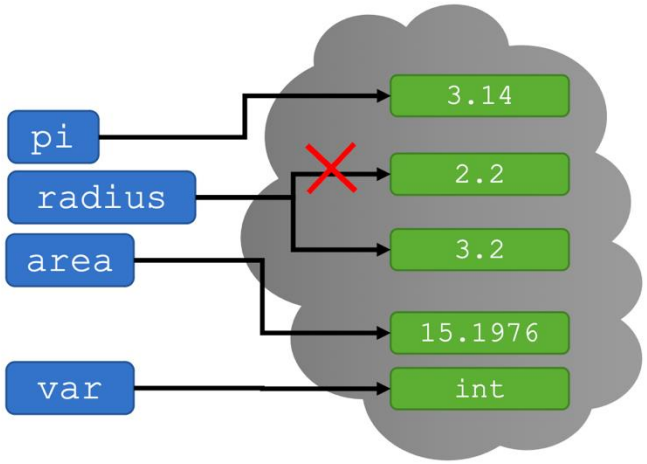
Today

- 字符串（STRINGS）
- 浮点数（FLOATS）
- 输入输出操作（INPUT/OUTPUT）
- 条件分支（CONDITIONS for BRUNCHING）

Python基础的总结

```
pi = 3.14
radius = 2.2
area = pi*(radius**2)
radius = radius+1

var = type(5*4)
```



■ 数据对象

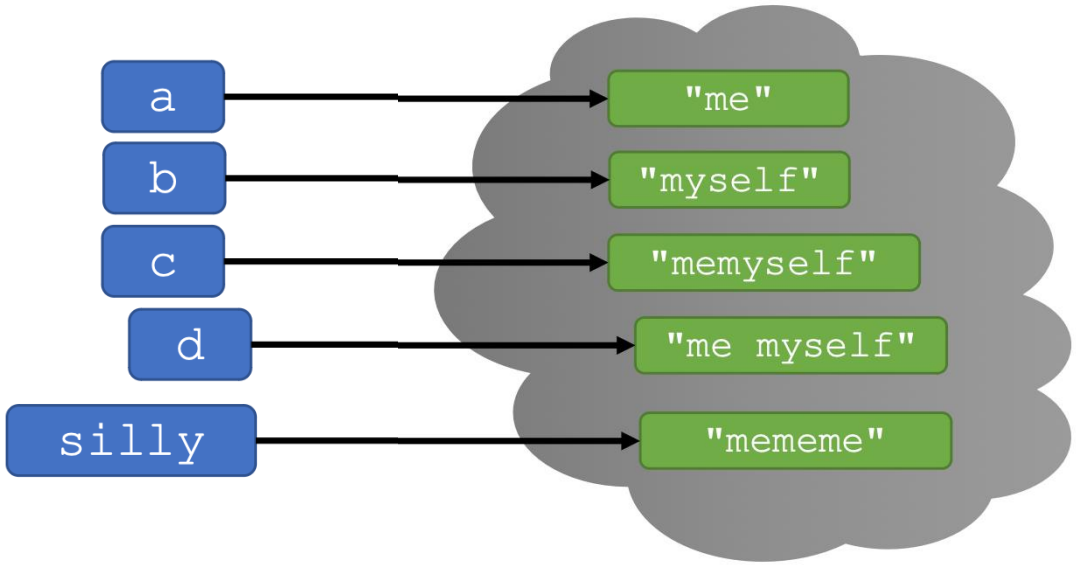
- ✓ 每个对象在计算机中拥有自己的类型
- ✓ 类型告诉Python可以对这个对象做哪些操作
- ✓ 表达式是对象和操作符的组合并对应着一个值
- ✓ 变量将数据对象赋予一个名字，“=”是赋值操作

■ 赋值程序

- ✓ 程序只执行你让它做的逻辑，程序的每一行按顺序执行
- ✓ 好的变量名称和注释可以帮助阅读代码

Chapter 1: STRINGS

- str（字符串）是Python程序中的数据类型，代表一串大小写敏感的字符序列
  - 字符：字母、数字、空格、特殊字符等
- Python字符串的写法是将字符串包裹在双引号（“”）或单引号（‘’）之间
  - 注意符号的一致性：
  - a = “me”
  - z = ‘you’
- 字符串有丰富的操作，这让它非常有用！
  - b = “myself”
  - c = a + b
  - d = a + “ ” + b
  - silly = a * 3



- 自己尝试下，s1和s2的值分别是什么?

```
b = ":"
c = ")"
s1 = b + 2 * c
f = "a"
g = " b"
h = "3"
s2 = (f + g) * int(h)
```

■ 字符串常用操作之——长度计算

- len( )：用于获取括号中字符串变量的字符串长度
  - s = “abc”
  - s_len = len(s) # is 3
- 越长的字符串，len会不会执行越慢?
  - 不会，字符串的长度在创建数据对象时就已确定
  - 由底层C语言代码高效计算出来，存为该字符串对象的属性
  - **len() 的本质是调用对象的内置方法**

■ 字符串常用操作之——字符串截取

- 截取单个字符——indexing
  - 方括号用于对字符串做索引来得到特定位置的字符（还是str类型）
  - s = “abc”
  - 正向索引从0开始，逆向索引从-1开始

```
s[0]  #  “a”
s[1]  #  “b”
s[-1]  #  “c”
s[-2]  #  “b”
s[3] # IndexError: string index out of range
```

■ 字符串常用操作之——字符串截取

- 截取字符子串——SUBSTRING
  - 使用 [start : stop : step] 来截取字符串

    从start开始到stop-1结束，每隔step取一个
  - 如果只用 [start : stop]， 默认step=1
  - 如果只用 [start :]，默认从start开始到字符串结尾
  - 如果只用 [:]，默认从字符串开头到结尾
  - 如果只用 [: end]，？
  - 如果step是负数，？

■ 字符串常用操作之——字符串截取

- 字符串截取例子

s= “abcdefgh”

index: 0 1 2 3 4 5 6 7
index: -8 -7 -6 -5 -4 -3 -2 -1

- s[3:6] # 得到“def”跟 s[3:6:1] 一样
- s[3:6:2] #得到 “df”
- s[:] # 得到 “abcdefgh”，跟 s[0:len(s):1] 一样
- s[::-1] # 得到 “hgfedcba”
- s[4:1:-2] # 得到 “ec”

- 自己尝试以下字符串截取的结果

```
s = "ABC d3f ghi"

s[3:len(s)-1]

s[4:0:-1]

s[6:3]
```

■ 字符串类型还有很多操作

```
>>> dir(x)
['__add__', '__class__', '__contains__', '__delattr__', '__dir__', '__doc_
_', '__eq__', '__format__', '__ge__', '__getattribute__', '__getitem__', '
__getnewargs__', '__gt__', '__hash__', '__init__', '__init_subclass__', '_
_iter__', '__le__', '__len__', '__lt__', '__mod__', '__mul__', '__ne__', '
__new__', '__reduce__', '__reduce_ex__', '__repr__', '__rmod__', '__rmul__
', '__setattr__', '__sizeof__', '__str__', '__subclasshook__', 'capitalize
', 'casefold', 'center', 'count', 'encode', 'endswith', 'expandtabs', 'fin
d', 'format', 'format_map', 'index', 'isalnum', 'isalpha', 'isascii', 'isd
ecimal', 'isdigit', 'isidentifier', 'islower', 'isnumeric', 'isprintable',
 'isspace', 'istitle', 'isupper', 'join', 'ljust', 'lower', 'lstrip', 'mak
etrans', 'partition', 'removeprefix', 'removesuffix', 'replace', 'rfind',
'rindex', 'rjust', 'rpartition', 'rsplit', 'rstrip', 'split', 'splitlines'
, 'startswith', 'strip', 'swapcase', 'title', 'translate', 'upper', 'zfill
```

- 用dir( )语句，可以查看一个对象的内置方法
- 自己下去多尝试，这些主要的方法作用效果是什么？

- “不可变”的STRING
  - 字符串在Python中是不可变的（immutable）
  - 可以通过赋值语句创建新的字符串对象给同一个变量（对象替换）
  - 不可以直接在原有字符串上进行修改（原位替换）
  - 如：
    - s = “car”
    - s[0] = ‘b’ # TypeError: 'str' object does not support item assignment
    - s = ‘b’ + s[1 : len(s)] # 正确，s变量被赋予新的对象“bar”

- 高效学习Python语法的方式：马上去控制台尝试输出的内容

Chapter 2: FLOATS

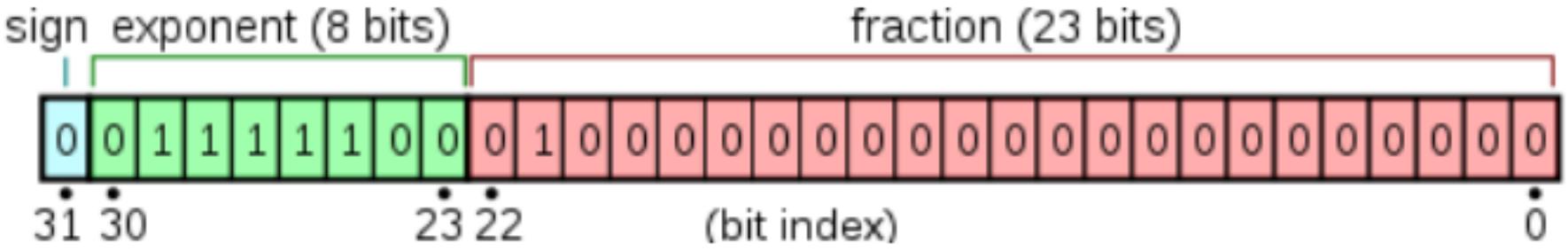
■ 浮点数

- 为什么叫作“浮点数”？
- 浮点数被称为“浮点数”（floating-point number）的原因与其在计算机中的表示方式密切相关。这一名称来源于数值的小数点位置可以“浮动”（即动态调整），以灵活表示极大或极小的数值范围。

- 浮点数
  - 灵感来自科学计数法：
    - 6.022e23 或 1.6e-19
    - 尾数（如 6.022 或 1.6）表示有效数字
    - 指数（如23 或 -19）表示数量级
    - 调整指数，小数点可以“浮动”到不同位置，高效覆盖广泛的数字范围
      - 1.23 × 10^4 = 12300（小数点右移4位）
      - 1.23 × 10^-2 = 0.0123（小数点左移2位）

- 浮点数
  - 计算机为什么要用浮点数?
  - 兼顾“大范围”和“小精度”
  - 通过动态调整指数，使小数点位置“浮动”，从而在有限的存储空间中兼顾大范围和小精度（例如 3.14 和 6.022e23 使用相同的存储格式）。

二进制存储



符号位　　指数位（偏移量）　　尾数位

■ 浮点数

- 一些浮点数换算的例子：
  - (1, 1) → $1*2^1$ → 10（2进制） → 2.0
  - (1, -1) → $1*2^{-1}$ → 0.1（2进制） → 0.5
  - (125, -2) → $125*2^{-2}$ → 11111.01（2进制） → 31.25
    - 125的二进制是1111101
    - 向左浮动2位得到11111.01
    - （二进制中乘以2的几次幂就是移动小数点几次，正数次幂向右，负数次幂向左）
    - 转十进制得到31.25

■ 浮点数运算的有趣现象

- 0.1 + 0.2 == 0.3 # False
- 0.1 + 0.2 # 0.30000000000000004

为什么?

➢ 精度丢失

计算机运算时会将输入的十进制转换为二进制，很多数字会落入无限循环的小数位，但计算机存不下那么多小数（32位），因此会进行截取，导致精度产生损失，计算结果只是个近似值。

对于大部分场景损失很小的精度无关紧要，也可使用高精度运算（64位）

■ 自己尝试运行以下代码并观察结果：

例子1:

```
x = 0
for i in range(10):
    x += 0.125
print(x == 1.25)
```

例子2:

```
x = 0
for i in range(10):
    x += 0.1
print(x == 1)
```

循环10次，累加一个浮点数到一个变量x中，判断变量是否等于预期的浮点数值

■ 浮点数运算的注意事项

- 永远不要用==判断两个浮点数是否相等
  1. 设定误差范围：a-b < 1e-9
  2. 现成的容差函数：math.isclose(0.1 + 0.2, 0.3)
  3. 限制小数位数比较：round(0.1 + 0.2, 5) == round(0.3, 5)
- 要小心设计使用浮点数的算法代码
  1. 数值超过浮点数能表示的最大范围：1e308 * 10 # inf
  2. 数值过小接近零被四舍五入：1e-324 * 0.1 # 0.0
  3. 将整数转换为浮点数丢失精度：float(1234567890123456789) # 1.2345678901234567e+18 （损失精度）

Chapter 3: INPUT/OUTPUT

- 打印语句（print）
  - 用于在命令行中输出内容
    - 在交互式控制台：可以直接输出表达式结果

```
>>> 3+2
5
```

    - 在.py文件：需要print语句才能在执行程序时输出结果

```
python output.py
5
```

```
print(3+2)
```

  - 在一个命令中打印多个对象信息
    - 用**逗号**分隔要打印的对象，输出以**空格**分隔的结果
    - 用**加号**拼接字符串打印一个整体的结果

```
a = "the"
b = 3
c = "musketeers"
print(a, b, c) # "the 3 musketeers"
print(a + str(b) + c) # "the3musketeers"
```

- 输入语句（input）
  - 有时，需要程序支撑用户的在线输入，并用输入的内容做灵活的处理
  - `x = input("Your input text: ")`
    - 用户可以在“Your input text: ”后面输入任何字符串
    - 例如输入“hi”并按回车，这个字符串作为input的返回值被赋给x变量
    - print(5*text) # hihihihihi

- 输入语句（input）
  - 需要注意的是，input总是返回字符串类型（str）
  - 如果要处理数字，需要转换对象类型
    - num1 = input(“Type a number: ”) # type 3
    - print(5*num1) # 33333
    - num2 = int(input(“Type a number: ”)) # type 3
    - print(5*num2) # 15

- **自己尝试实现以下要求的输出效果：**
  - 让用户输入一个动词
  - 1. 打印“I can __ better than you!”，替换 __为你让用户输入的动词
  - 2. 在一行中打印这个动词5次，以空格分隔每两个动词
  - 举例：
    - 如果用户输入的是“run”
    - 你打印出：

```
I can run better than you!
run run run run run
```

■ 格式化字符串输出

- 从Python 3.6之后支持
- 任何的操作可以通过一个字符串表示
- 操作的表达式由花括号{ }包裹
- 表达式在运行语句时被自动计算，并转换为字符串类型，再与外面的字符串拼接起来

```
>>> num = 300
>>> fraction = 1/3
>>> print(num*fraction, 'is', fraction*100, '% of', num)
100.0 is 33.33333333333333 % of 300
>>> print(num*fraction, 'is', str(fraction*100) + '% of', num)
100.0 is 33.33333333333333% of 300
>>> print(f'{num*fraction} is {fraction*100}% of {num}')
100.0 is 33.33333333333333% of 300
>>>
```

用逗号时以空格分隔

同时用逗号和拼接操作，拼接没有空格

用格式化字符串输出，{}中的表达式自动执行

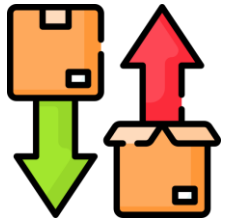
- 表达式可以被置于任何位置!
- Python会在执行时自动计算表达的值

```
>>> num = 300
>>> fraction = 1/3
>>> print(num*fraction, 'is', fraction*100, '% of', num)
100.0 is 33.333333333333336 % of 300
>>> print(num*fraction, 'is', str(fraction*100) + '% of', num)
100.0 is 33.333333333333336% of 300
>>> print(f'{num*fraction} is {fraction*100}% of {num}')
100.0 is 33.333333333333336% of 300
>>>
```

Chapter 4: CONDITIONS for BRANCHING

- 赋值与逻辑相等
  - 计算机中的两种“相等”：

1. variable = value
   - 将变量variable的值赋为value
2. expression1 == expression2
   - 测试是否“相等”
   - 不发生赋值操作
   - 整个语句可以被True或False代替



- 比较操作（Comparison）
  - 把变量i与变量j进行比较，int, float, string等类型的变量都可以比较
  - 比较操作的结果是布尔类型的值（Boolean），只有True和False两个值

```
i > j
i >= j
i < j
i <= j
i == j
i != j
```

注意：避免直接比较小数

```
>>> 5 == 5.0
True
>>> 0.1+0.2 == 0.3
False
```

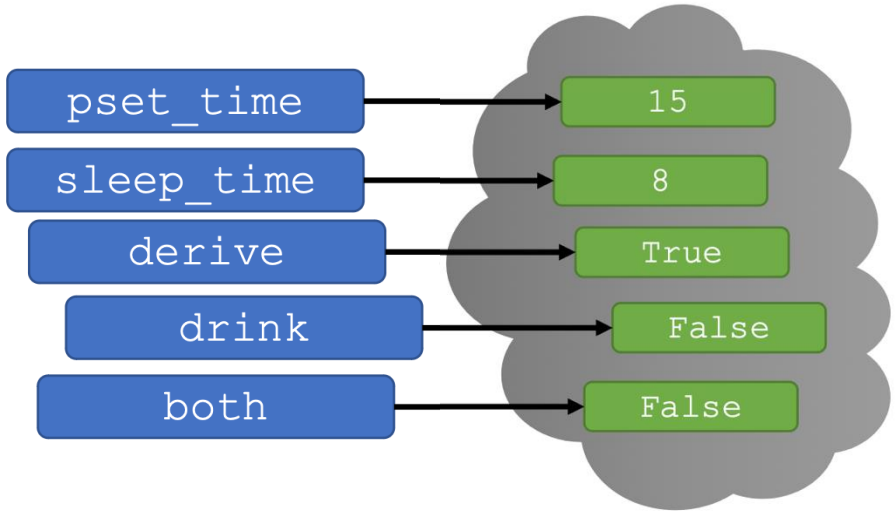
- 布尔逻辑运算（Logical Operators on bool）
  - 设a和b为具有布尔类型的变量名（值为True或False）
  - `not a # True` 如果 `a` 为 `False`
  - `a and b # True` 如果 `a` 和 `b` 都为 `True`
  - `a or b # True` 如果 `a` 和 `b` 至少有一个为 `True`

A	B	A and B	A or B
True	True	True	True
True	False	False	True
False	True	False	True
False	False	False	False

CONDITIONS for BRANCHING

- 一些比较操作的例子

```
pset_time = 15
sleep_time = 8
print(sleep_time > pset_time) # False
derive = True
drink = False
both = drink and derive
print(both) # False
```



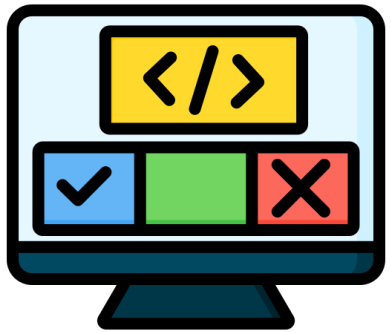
- 常用比较操作一览

Operator	Meaning	Example	Value
==	equals	`1 + 1 == 2`	`True`
!=	does not equal	`3.2 != 2.5`	`True`
<	less than	`10 < 5`	`False`
>	greater than	`10 > 5`	`True`
<=	less than or equal to	`126 <= 100`	`False`
>=	greater than or equal to	`5.0 >= 5.0`	`True`

- **自己尝试写一段程序**：
  - 设置一个秘密的数字，赋给一个变量
  - 让用户猜一个数字输入进来
  - 根据所猜的数字与你的秘密数字是否相等，打印出布尔值False或True

CONDITIONS for BRANCHING

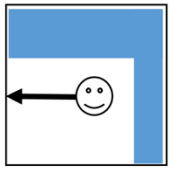
- 我们为什么需要布尔类型的值?
  - 当我们要对程序进行流程控制时，我们可以基于布尔变量的结果来设计不同的代码逻辑
  - “如果某条件满足（True），执行A，否则（False），执行B”



- 基于布尔逻辑运算的例子

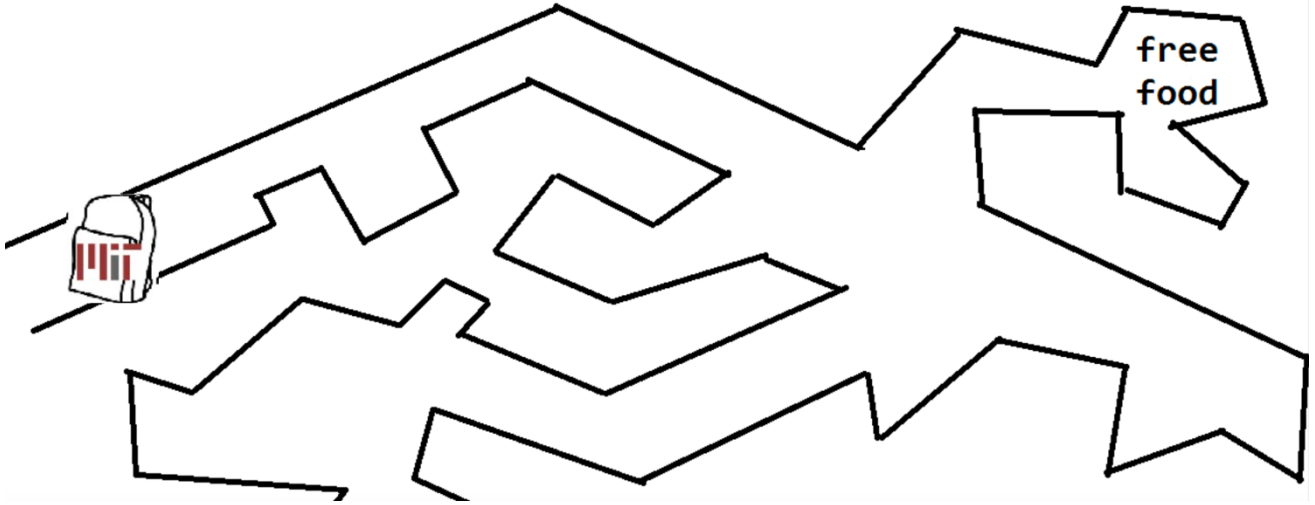
If right clear, go right

If right blocked, go forward



If right and front blocked, go left

If right , front, left blocked, go back



基于条件做流程控制

- 以上我们介绍了逻辑运算和流程控制，Python中怎么写条件流程控制呢?

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

- <condition>会产生布尔类型的值（True 或者 False）
- 注意Python程序中的缩进（四个\s或者一个\t）（有些语言不care）
- 如果条件满足（True），就执行if下面的<code>，否则执行else下面的

- 基于条件做流程控制
  - 以上我们介绍了逻辑运算和流程控制，Python中怎么写条件流程控制呢?

```
if condition:
    statements          # note indenting
else:
    statements          # note indenting
```

if-else 句法

```
def invert_beepers():
    if beepers_present():
        pick_beeper() # note indenting
    else:
        put_beeper()  # note indenting
```

if-else 函数例子

CONDITIONS for BRANCHING

- 基于条件做流程控制

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

- 这段代码是如何执行的?
- 按顺序，遇到第一个<condition>为True的，就执行它下面的代码

CONDITIONS for BRANCHING

- 基于条件做流程控制

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

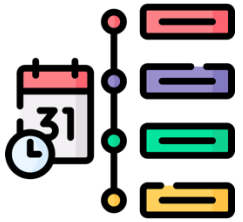
  - 这段代码是如何执行的?
  - 遇到第一个<condition>为True的，就执行它下面的代码
  - 如果没有<condition>为True，就执行else下面的代码

CONDITIONS for BRANCHING

■ 基于条件做流程控制

- 一个例子：根据一天的工作和睡觉小时数来输出不同的信息

```
work_time = ???
sleep_time = ???
if (work_time + sleep_time) > 24:
      print("Impossible!")
elif (work_time + sleep_time) == 24:
      print("Full schedule!")
else:
      leftover = 24 - work_time - sleep_time
      print(leftover, "h of free time!")
print("end of day")
```



- 基于条件做流程控制
  - 更多的例子

```
if 1 < 2 :
   print("1 is less than 2")
```

```
num = int(input("Enter a number: "))
if num == 0:
   print("That number is 0")
else :
   print("That number is not 0.")
```

- 基于条件做流程控制
  - 更多的例子

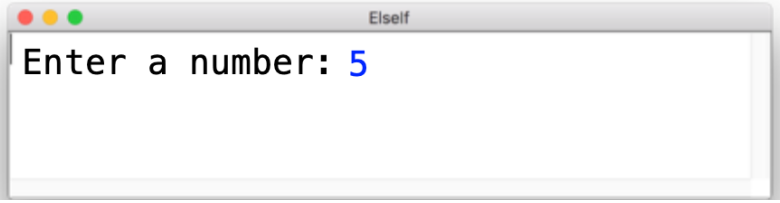
```
num = int(input("Enter a number: "))
if num == 0:
   print("Your number is 0 ")
else:
   if num > 0:
      print("Your number is positive")
   else:
      print("Your number is negative")
```

- 基于条件做流程控制
  - 更多的例子

```
num = int(input("Enter a number: "))
if num == 0:
    print("Your number is 0 ")
elif num > 0:
    print("Your number is positive")
else:
    print("Your number is negative")
```

- 基于条件做流程控制
  - 更多的例子


```
num = int(input("Enter a number: "))
if num == 0:
   print("Your number is 0 ")
elif num > 0:
   print("Your number is positive")
else:
   print("Your number is negative")
```

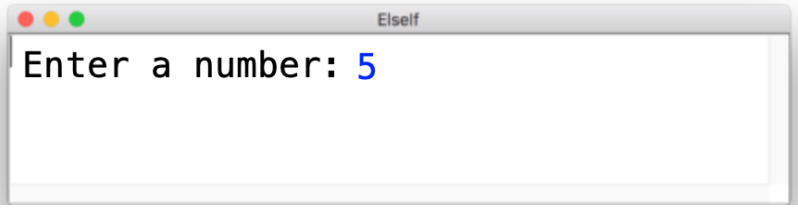


基于条件做流程控制

- 更多的例子

5

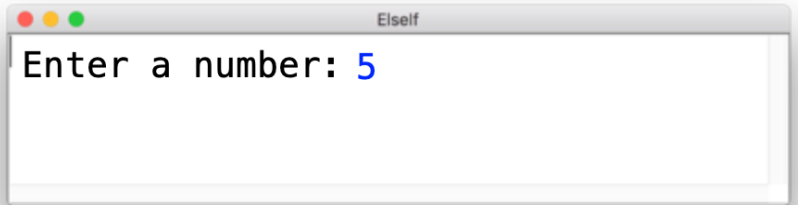
```
num = int(input("Enter a number: "))
if num == 0:
   print("Your number is 0 ")
elif num > 0:
   print("Your number is positive")
else:
   print("Your number is negative")
```

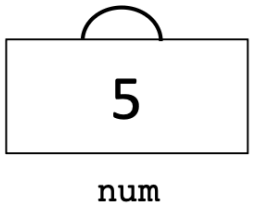


基于条件做流程控制

- 更多的例子

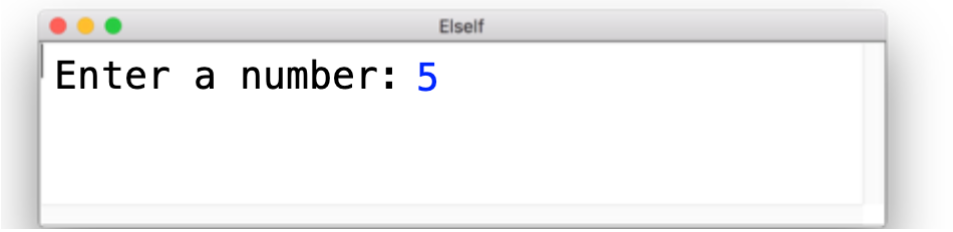

```
num = int(input("Enter a number: "))
if num == 0:
   print("Your number is 0 ")
elif num > 0:
   print("Your number is positive")
else:
   print("Your number is negative")
```

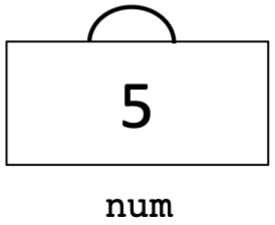




- 基于条件做流程控制
  - 更多的例子

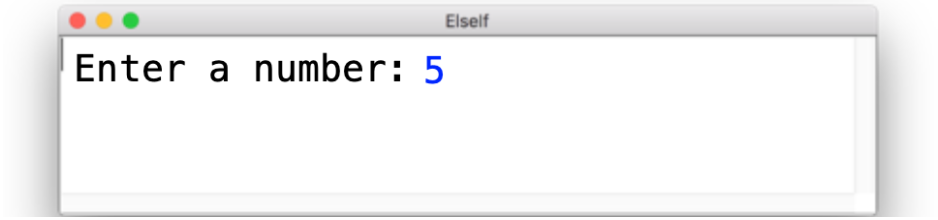
```
num = int(input("Enter a number: "))
if num == 0:
    print("Your number is 0 ")
elif num > 0:
    print("Your number is positive")
else:
    print("Your number is negative")
```

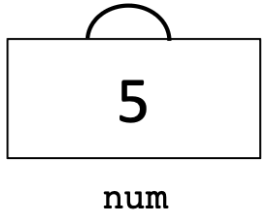




- 基于条件做流程控制
  - 更多的例子

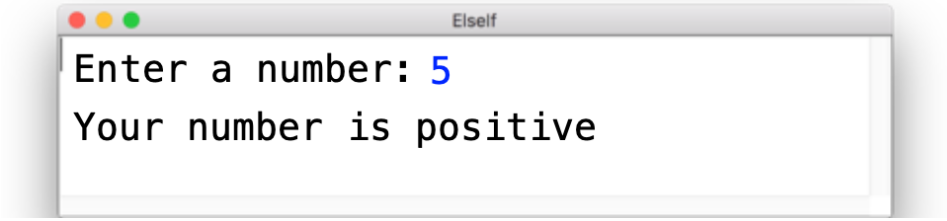
```
num = int(input("Enter a number: "))
if num == 0:
   print("Your number is 0 ")
elif num > 0:
   print("Your number is positive")
else:
   print("Your number is negative")
```

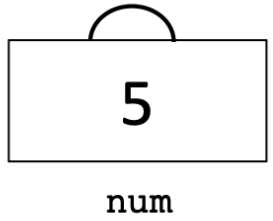




- 基于条件做流程控制
  - 更多的例子

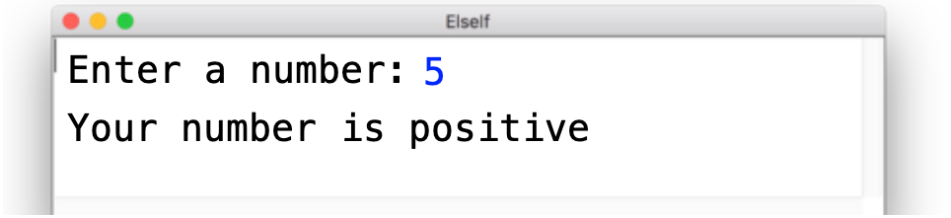
```
num = int(input("Enter a number: "))
if num == 0:
   print("Your number is 0 ")
elif num > 0:
   print("Your number is positive")
else:
   print("Your number is negative")
```

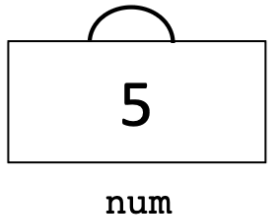




- 基于条件做流程控制
  - 更多的例子

```
num = int(input("Enter a number: "))
if num == 0:
   print("Your number is 0 ")
elif num > 0:
   print("Your number is positive")
else:
   print("Your number is negative")
```





自己尝试修改以下代码使它执行正确的逻辑

- 提示：注意缩进和逻辑性

```
x = int(input("Enter a number for x: "))
y = int(input("Enter a different number for y: "))
if x == y:
   print(x,"is the same as",y)
print("These are equal!")
elif x > y:
   print(x,"is bigger than",y)
print(x,"is smaller than",y)
```

■ 缩进和嵌套逻辑分支

- Python中的缩进很重要，具有逻辑语义，直接影响代码的正确性

```
if x == y:
    print("x and y are equal")
    if y != 0:
        print("therefore, x / y is", x/y)
elif x < y:
    print("x is smaller")
else:
    print("y is smaller")
print("thanks!")
```

- 上下缩进对齐的代码，在逻辑上具有并列性（按顺序依次执行）

- **自己尝试下，如果把下面的 `if x >= y` 改为 `elif x >= y`，会输出什么?**

```
answer = ''
x = 11
y = 11
if x == y:
    answer = answer + 'M'
if x >= y:
    answer = answer + 'i'
else:
    answer = answer + 'T'
print(answer)
```

- **自己尝试编写一个程序，实现以下功能：**
  - 设置一个秘密的数字，赋给一个变量
  - 让用户猜一个数字输入进来 （跟之前一样）
  - 设置条件流程控制，输出所猜的数字是否太小、太大、或跟猜的一样

Chapter 5: SUMMARY

SUMMARY

- 我们学习了字符串（Strings）这种数据类型
  - 字符串是一串字符序列，第一个字符的索引是0
  - 字符串可以通过索引进行截取（slicing），获取子串
- 我们学习了浮点数（Float）这种计算机中的小数
  - 其设计灵感来自科学计数法中小数点的浮动
  - 通过小数点位置“浮动” 兼顾大范围和小精度

SUMMARY

- 我们学习了输入输出（Input/Output）这种程序交互方法
  - 输入由`input`命令完成，任何用户输入的内容被读取为字符串类型
  - 输出由`print`命令完成，只有在.py文件中`print`出来的内容才能在终端中看到
- 我们学习了条件流程控制（条件分支）这种逻辑编写方法
  - 当条件输出结果为`True`时，程序会执行其下面的代码内容
  - 在`if-elif-elif-…`的程序结构中，第一个条件为`True`下的代码会被执行
  - 在Python中，代码的缩进很重要，直接影响代码执行逻辑

Reading and QA Time

See you next week !