# Lecture 1 第一讲



## Conditionals 条件

- Conditionals allow you, the programmer, to allow your program to make decisions: As if your program has the choice between taking the left-hand road or the right-hand road based upon certain conditions.

  条件允许您（程序员）允许您的程序做出决定：就好像您的程序可以根据某些条件选择走左路或右路一样。
- Built within Python are a set of "operators" that are used to ask mathematical questions.

  Python 中内置了一组用于提出数学问题的"运算符"。
- `>` and `<` symbols are probably quite familiar to you.

  `>` 和 `<` 符号您可能非常熟悉。
- `>=` denotes "greater than or equal to."

  `>=` 表示"大于或等于"。
- `<=` denotes "less than or equal to."

  `<=` 表示"小于或等于"。
- `==` denotes "equals, though do notice the double equal sign! A single equal sign would assign a value. Double equal signs are used to compare variables.

  `==` 表示"等于，但请注意双等号！单个等号将分配一个值。双等号用于比较变量。
- `!=` denotes "not equal to.

  `!=` 表示"不等于。
- Conditional statements compare a left-hand term to a right-hand term.

  条件语句将左侧术语与右侧术语进行比较。

## if Statements if 语句

- In your terminal window, type `code compare.py`. This will create a brand new file called "compare."

  在终端窗口中，键入 `code compare.py`。这将创建一个名为"compare"的全新文件。
- In the text editor window, begin with the following:

  在文本编辑器窗口中，从以下内容开始：

```
x = int(input("What's x? "))
y = int(input("What's y? "))

if x < y:
    print("x is less than y")
```

  Notice how your program takes the input of the user for both x and y, casting them as integers and saving them into their respective x and y variables. Then, the `if` statement compares x and y. If the condition of `x < y` is met, the `print` statement is executed.

  请注意您的程序如何获取用户的 x 和 y 输入，将它们转换为整数并将它们保存到各自的 x 和 y 变量中。然后，`if` 语句比较 x 和 y。如果满足 `x < y` 的条件，则执行 `print` 语句。
- `if` statements use `bool` or boolean values (true or false) to decide whether or not to execute. If the statement of `x > y` is true, the compiler will register it as `true` and execute the code.

  `if` 语句使用 `bool` 或布尔值（true 或 false）来决定是否执行。如果 `x > y` 的语句为 true，编译器会将其注册为 `true` 并执行代码。

## Control Flow, elif, and else

## Control Flow、elif 和 else

- Further revise your code as follows:

  进一步修改您的代码，如下所示：

```
x = int(input("What's x? "))
y = int(input("What's y? "))

if x < y:
    print("x is less than y")
if x > y:
```

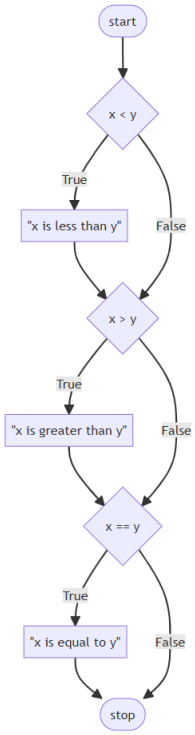
```
    print("x is greater than y")
if x == y:
    print("x is equal to y")
```

Notice how you are providing a series of `if` statements. First, the first `if` statement is evaluated. Then, the second `if` statement runs its evaluation. Finally, the last `if` statement runs its evaluation. This flow of decisions is called "control flow."

请注意您是如何提供一系列 `if` 语句的。首先，计算第一个 `if` 语句。然后，第二个 `if` 语句运行其评估。最后，最后一个 `if` 语句运行其评估。此决策流称为"控制流"。

- Our code can be represented as follows:

  我们的代码可以表示如下：




- This program can be improved by not asking three consecutive questions. After all, not all three questions can have an outcome of `true`! Revise your program as follows:

  可以通过不问三个连续的问题来改进此程序。毕竟，并非所有三个问题都可以获得 `true`！按如下方式修改您的程序：

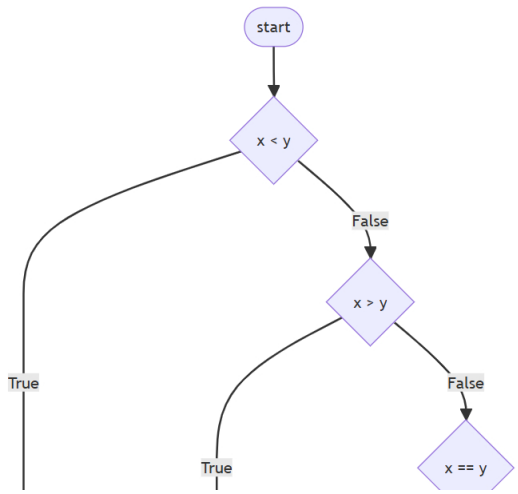
```
x = int(input("What's x? "))
y = int(input("What's y? "))

if x < y:
    print("x is less than y")
elif x > y:
    print("x is greater than y")
elif x == y:
    print("x is equal to y")
```

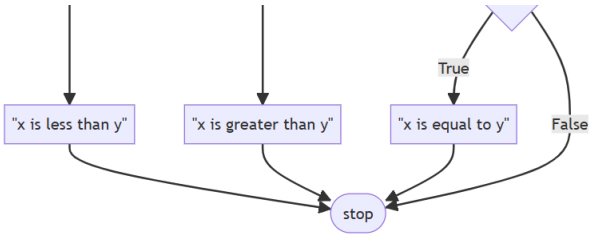
Notice how the use of `elif` allows the program to make fewer decisions. First, the `if` statement is evaluated. If this statement is found to be true, all the `elif` statements will not be run at all. However, if the `if` statement is evaluated and found to be false, the first `elif` will be evaluated. If this is true, it will not run the final evaluation.

请注意，使用 `elif` 如何使程序做出更少的决策。首先，评估 `if` 语句。如果发现此语句为 true，则根本不运行所有 `elif` 语句。但是，如果评估 `if` 语句并发现为 false，则将评估第一个 `elif`。如果为 true，则不会运行最终评估。

- Our code can be represented as follows:

  我们的代码可以表示如下：






- While your computer may not notice a difference speed-wise between our first program and this revised program, consider how an online server running billions or trillions of these types of calculations each day could definitely be impacted by such a small coding decision.

  虽然您的计算机可能没有注意到我们的第一个程序和这个修订后的程序在速度上的差异，但请考虑一下每天运行数十亿或数万亿个此类计算的在线服务器肯定会受到如此小的编码决策的影响。

- There is one final improvement we can make to our program. Notice how logically `elif x == y` is not a necessary evaluation to run. After all, if logically x is not less than y AND x is not greater than y, x MUST equal y. Therefore, we don't have to run `elif x == y`. We can create a "catch-all," default outcome using an `else` statement. We can revise as follows:

  我们可以对程序进行最后一项改进。请注意，从逻辑上讲 `elif x == y` 并不是运行所必需的计算。毕竟，如果逻辑上 x 不小于 y 且 x 不大于 y，则 x 必须等于 y。因此，我们不必运行 `elif x == y`。我们可以使用 `else` 语句创建一个 "catch-all" 默认结果。我们可以按如下方式进行修改：

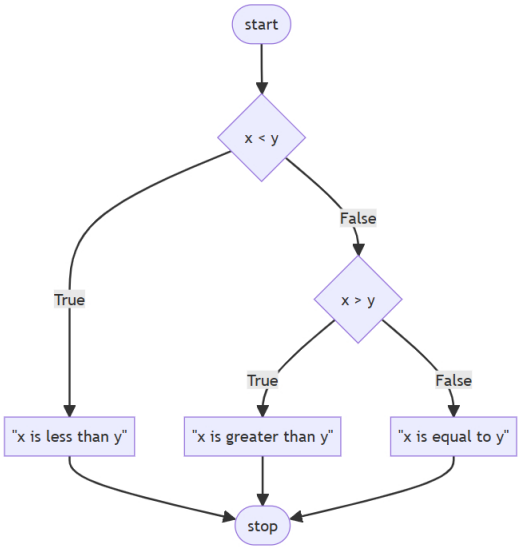
```
x = int(input("What's x? "))
y = int(input("What's y? "))

if x < y:
    print("x is less than y")
elif x > y:
    print("x is greater than y")
else:
    print("x is equal to y")
```

  Notice how the relative complexity of this program has decreased through our revision.

  请注意，在我们的修订版中，该程序的相对复杂性是如何降低的。

- Our code can be represented as follows:

  我们的代码可以表示如下：




## or 或

- `or` allows your program to decide between one or more alternatives. For example, we could further edit our program as follows:

  或允许您的程序在一个或多个备选方案之间进行选择。例如，我们可以进一步编辑我们的程序，如下所示：

```
x = int(input("What's x? "))
y = int(input("What's y? "))

if x < y or x > y:
    print("x is not equal to y")
else:
    print("x is equal to y")
```

  Notice that the result of our program is the same, but the complexity is decreased. The efficiency of our code is increased.

  请注意，我们的程序的结果是相同的，但复杂性降低了。我们的代码效率提高了。

- At this point, our code is pretty great. However, could the design be further improved? We could further edit our code as follows:

  在这一点上，我们的代码非常棒。但是，设计可以进一步改进吗？我们可以进一步编辑我们的代码，如下所示：

```
x = int(input("What's x? "))
y = int(input("What's y? "))

if x != y:
    print("x is not equal to y")
else:
    print("x is equal to y")
```

Notice how we removed the `or` entirely and simply asked, "Is x not equal to y?" We ask one and only one question. Very efficient!

请注意我们如何完全删除 `or`，并简单地问道："x 不等于 y 吗？我们问一个且只问一个问题。非常有效！

- For the purpose of illustration, we could also change our code as follows:

  为了便于说明，我们还可以按如下方式更改我们的代码：

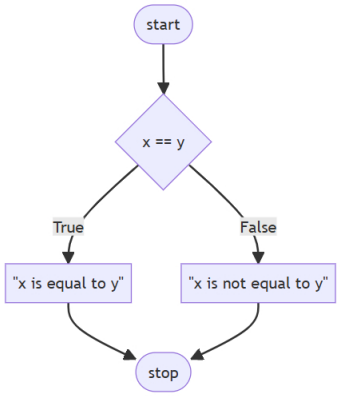
```
x = int(input("What's x? "))
y = int(input("What's y? "))

if x == y:
    print("x is equal to y")
else:
    print("x is not equal to y")
```

Notice that the `==` operator evaluates if what is on the left and right are equal to one another. The use of double equal signs is very important. If you use only one equal sign, an error will likely be thrown by the compiler.

请注意，`==` 运算符会评估左侧和右侧的内容是否彼此相等。使用双等号非常重要。如果只使用一个等号，编译器可能会引发错误。

- Our code can be illustrated as follows:

  我们的代码可以说明如下：




## and 和

- Similar to `or`, `and` can be used within conditional statements.

  与 `or` 类似，可以在条件语句中使用。

- Execute in the terminal window `code grade.py`. Start your new program as follows:

  在终端窗口中执行 `code grade.py`。按如下方式启动您的新程序：

```
score = int(input("Score: "))

if score >= 90 and score <= 100:
    print("Grade: A")
elif score >=80 and score < 90:
    print("Grade: B")
elif score >=70 and score < 80:
    print("Grade: C")
elif score >=60 and score < 70:
    print("Grade: D")
else:
    print("Grade: F")
```

Notice that by executing `python grade.py`, you will be able to input a score and get a grade. However, notice how there is potential for bugs.

请注意，通过执行 `python grade.py`，您将能够输入分数并获得等级。但是，请注意可能存在错误。

- Typically, we do not want to ever trust our users to input the correct information. We could improve our code as follows:

  通常，我们不想相信我们的用户会输入正确的信息。我们可以按如下方式改进我们的代码：

```
score = int(input("Score: "))

if 90 <= score <= 100:
    print("Grade: A")
elif 80 <= score < 90:
    print("Grade: B")
elif 70 <= score < 80:
    print("Grade: C")
elif 60 <= score < 70:
    print("Grade: D")
else:
    print("Grade: F")
```

Notice how Python allows you to chain together the operators and conditions in a way quite uncommon to other programming languages.

请注意 Python 如何允许您以其他编程语言非常不常见的方式将运算符和条件链接在一起。

- Still, we can further improve our program:

  尽管如此，我们还可以进一步改进我们的计划：

```
score = int(input("Score: "))

if score >= 90:
    print("Grade: A")
elif score >= 80:
    print("Grade: B")
elif score >= 70:
    print("Grade: C")
```

```
elif score >= 60:
    print("Grade: D")
else:
    print("Grade: F")
```

Notice how the program is improved by asking fewer questions. This makes our program easier to read and far more maintainable in the future.

请注意如何通过提出更少的问题来改进程序。这使得我们的程序更易于阅读，并且在将来更易于维护。

- You can learn more in Python's documentation on control flow.

  您可以在 Python 的 control flow 文档中了解更多信息。

## Modulo 模

- In mathematics, parity refers to whether a number is either even or odd.

  在数学中，奇偶校验是指一个数字是偶数还是奇数。

- The modulo `%` operator in programming allows one to see if two numbers divide evenly or divide and have a remainder.

  编程中的模 `%` 运算符允许查看两个数字是均匀除法还是除以并有余数。

- For example, 4 % 2 would result in zero, because it evenly divides. However, 3 % 2 does not divide evenly and would result in a number other than zero!

  例如，4 % 2 将产生零，因为它会均匀除数。但是，3 % 2 不会均匀除法，并且会产生一个非零的数字！

- In the terminal window, create a new program by typing `code parity.py`. In the text editor window, type your code as follows:

  在终端窗口中，通过键入 `代码 parity.py` 创建新程序。在文本编辑器窗口中，键入代码，如下所示：

```
x = int(input("What's x? "))

if x % 2 == 0:
    print("Even")
else:
    print("Odd")
```

Notice how our users can type in any number 1 or greater to see if it is even or odd.

请注意我们的用户如何键入任何数字 1 或更大的数字来查看它是偶数还是奇数。

## Creating Our Own Parity Function

## 创建我们自己的奇偶校验函数

- As discussed in Lecture 0, you will find it useful to create a function of your own!

  正如第 0 讲中所讨论的，你会发现创建自己的函数很有用！

- We can create our own function to check whether a number is even or odd. Adjust your code as follows:

  我们可以创建自己的函数来检查数字是偶数还是奇数。按如下方式调整您的代码：

```
def main():
    x = int(input("What's x? "))
    if is_even(x):
        print("Even")
    else:
        print("Odd")


def is_even(n):
    if n % 2 == 0:
        return True
    else:
        return False


main()
```

Notice that our `if` statement `is_even(x)` works even though there is no operator there. This is because our function returns a `bool` (or boolean), true or false, back to the main function. The `if` statement simply evaluates whether or not `is_even` of `x` is true or false.

请注意，即使那里没有运算符，我们的 `if` 语句 `is_even（x）` 也能正常工作。这是因为我们的函数将 `bool` （或布尔值）返回给主函数。`if` 语句只是评估 `x` 的 `is_even` 是 true 还是 false。

## Pythonic

- In the programming world, there are types of programming that are called "Pythonic" in nature. That is, there are ways to program that are sometimes only seen in Python programming. Consider the following revision to our program:

  在编程世界中，有一些类型的编程本质上称为 "Pythonic"。也就是说，有些编程方法有时只能在 Python 编程中看到。请考虑对我们的程序进行以下修订：

```
def main():
    x = int(input("What's x? "))
    if is_even(x):
        print("Even")
    else:
        print("Odd")


def is_even(n):
    return True if n % 2 == 0 else False


main()
```

Notice that this return statement in our code is almost like a sentence in English. This is a unique way of coding only seen in Python.

请注意，我们代码中的这个 return 语句几乎就像英语中的句子。这是一种只有在 Python 中才能看到的独特编码方式。

- We can further revise our code and make it more and more readable:

  我们可以进一步修改我们的代码，使其越来越可读：

```
def main():
    x = int(input("What's x? "))
    if is_even(x):
        print("Even")
    else:
        print("Odd")


def is_even(n):
    return n % 2 == 0


main()
```

  Notice that the program will evaluate what is happening within the `n % 2 == 0` as either true or false and simply return that to the main function.

  请注意，程序会将 `n % 2 == 0` 内发生的事情评估为 true 或 false，然后简单地将其返回给 main 函数。

## `match` 火柴

- Similar to `if`, `elif`, and `else` statements, `match` statements can be used to conditionally run code that matches certain values.

  与 `if`、`elif` 和 `else` 语句类似，`match` 语句可用于有条件地运行与某些值匹配的代码。

- Consider the following program:

  请考虑以下程序：

```
name = input("What's your name? ")

if name == "Harry":
    print("Gryffindor")
elif name == "Hermione":
    print("Gryffindor")
elif name == "Ron":
    print("Gryffindor")
elif name == "Draco":
    print("Slytherin")
else:
    print("Who?")
```

  Notice the first three conditional statements print the same response.

  请注意，前三个条件语句打印相同的响应。

- We can improve this code slightly with the use of the `or` keyword:

  我们可以通过使用 `or` 关键字来稍微改进这段代码：

```
name = input("What's your name? ")

if name == "Harry" or name == "Hermione" or name == "Ron":
    print("Gryffindor")
elif name == "Draco":
    print("Slytherin")
else:
    print("Who?")
```

  Notice the number of `elif` statements has decreased, improving the readability of our code.

  请注意，`elif` 语句的数量减少了，从而提高了代码的可读性。

- Alternatively, we can use `match` statements to map names to houses. Consider the following code:

  或者，我们可以使用 `match` 语句将名称映射到房屋。请考虑以下代码：

```
name = input("What's your name? ")

match name:
    case "Harry":
        print("Gryffindor")
    case "Hermione":
        print("Gryffindor")
    case "Ron":
        print("Gryffindor")
    case "Draco":
        print("Slytherin")
    case _:
        print("Who?")
```

  Notice the use of the `_` symbol in the last case. This will match with any input, resulting in similar behavior as an `else` statement.

  请注意在最后一种情况下使用了 `_` 符号。这将与任何输入匹配，从而产生与 `else` 语句类似的行为。

- A match statement compares the value following the `match` keyword with each of the values following the `case` keywords. In the event a match is found, the respective indented code section is executed, and the program stops the matching.

  match 语句将 `match` 关键字后面的值与 `case` 关键字后面的每个值进行比较。如果找到匹配项，则执行相应的缩进代码部分，程序将停止匹配。

- We can improve the code:

  我们可以改进代码：

```
name = input("What's your name? ")

match name:
    case "Harry" | "Hermione" | "Ron":
        print("Gryffindor")
    case "Draco":
        print("Slytherin")
    case _:
        print("Who?")
```

  Notice, the use of the single vertical bar `|`. Much like the `or` keyword, this allows us to check for multiple values in the same `case` statement.

请注意，使用了单个竖线 `|`。与 `or` 关键字非常相似，这允许我们检查同一 `case` 语句中的多个值。

## Summing Up 总结

You now have the power within Python to use conditional statements to ask questions and have your program take action accordingly. In this lecture, we discussed...

现在，您可以在 Python 中使用条件语句来提出问题，并让您的程序相应地采取行动。在本次讲座中，我们讨论了...

- Conditionals; 条件;
- `if` Statements; `如果` 语句;
- Control flow, `elif`, and `else`;
  Control flow、`elif` 和 `else`;
- `or`; `或者`;
- `and`; `和`;
- Modulo; 模;
- Creating your own function;
  创建您自己的函数;
- Pythonic coding; Pythonic 编码;
- and `match`. 和 `match` 进行匹配。