1 Truth

Last lecture we learned about the `int`, `float`, and `string` types. Another very important object type in Python is the **boolean** type. The two reserved keywords `True` and `False` are values with type boolean.

Expressions That Can Evaluate to Boolean Values

comparison operators We can compare objects to test *order* ($<$, $>$, $<=$, $>=$) or *equality* ($==$, $!=$). Order comparisons are not defined for all objects.

identity operators Identity is slightly different than equality, but identity is tested with the `is` and `is not` keywords. The expression `x is y` yields `True` only if `x` and `y` are the same object.

membership Some Python objects contain other objects (later we will discuss `range` objects). For these objects, the `in` and `not in` keywords can be used to test for object membership.

variables Variables refer to values, so variables can refer to boolean values.

logical operators The `and`, `or`, and `not` keywords are Python's logical operators. The truth table below shows the result of these operators on different values.

A	**B**	**A** `and` **B**	**A** `or` **B**	`not` **A**
False	False	False	False	True
False	True	False	True	True
True	False	False	True	False
True	True	True	True	False

In addition, objects evaluate to `True` or `False` in boolean contexts:

- `False`, `None`, all values of zero, the empty string, and empty containers all evaluate to `False`
- Everything else evaluates to `True`

2 Conditional Statements

Together with boolean values, we can use the `if`/`elif`/`else` keywords to affect the *control flow* of our programs. In other words we can cause certain statements to be executed *conditionally*.

```
1  a = True
2  if a :
3      print("variable a is true")
4  else :
5      print("variable a is false")
```

When the Python interpreter reaches line 2, the '`if`' checks the value of its expression (in this example, the expression is just the variable `a`). The interpreter executes all of the indented code between an `if` and `else` when the expression yields `True`. Otherwise, that region of code is skipped, and the interpreter executes the indented region of code following the `else` keyword.

This example illustrates a very important fact about Python: indentation matters! Python uses indentation as a signal that a region of text is a group (all consecutive text at the same indentation level).

There may be any number of `elif`s between an `if` and `else`. The Python interpreter will continue to evaluate the boolean expressions after each `if` and `elif` keyword, in order, until one of those expressions yields `True`. Otherwise only the `else` section is executed.

Finally, it is not necessary to have an `else` statement. For example,

```
1 want_pepperoni = False
2 print("I want to order a ", end='')
3 if want_pepperoni :
4     print("pepperoni ")
5 print("pizza!")
```

will print "I want to order a pizza!" Since there is not else statement, the Python interpreter simply skips the indented code and continues to execute the code at the same indendation level as the original `if` expression.

3 Loops

Loops can be used to execute a sequence of expressions repeatedly. There are two main types of loops, the `while` loop and the `for` loop. The `while` loop continues to execute a sequence of expressions as long as its condition remains `True`. The `for` loop iterates over a sequence of objects.

3.1 `while`

Here's some code that prints the prime factors of the integer `n` from largest to smallest. Notice that the `while` expression evaluates boolean expression; if the expressions true, it executes the body and then repeats.

```
1  n = 99
2  i = n
3  while (i > 0):
4      if (n % i == 0):
5          print(i)
6      i = i − 1
```

```
$ python3 factors.py
99
33
11
9
3
1
```

We can also use the `break` keyword to leave a loop at any time:

```
1  bottles = 99
2  enough_already = 42
3  while (bottles > 0):
4      print(str(bottles) + " bottles of beer on the wall,")
5      print(str(bottles) + " bottles of beer!")
6      print("Take one down, pass it around...")
7      bottles = bottles − 1
8      print(str(bottles) + " bottles of beer on the wall!\n")
9      ## Has anyone ever actually finished this song?
10     if (bottles == enough_already):
11         print("I can't take anymore of this!")
12         break
```

I think we all know how that output might look. . .

3.2 `for` and `range`

The keyword `for` is used to iterate over sequences of objects. We actually seen one type of sequence already—strings.

```
1 for i in "this sentence is false"
2     print(i,end=" ")
3 print()
```

Notice here we make use of the optional keyword argument to print `end`, which says how the string should be terminated (by default `print()` ends with a newline '`\n`'. Thus, the function `print()` just prints a newline character.).

The `for` keyword is often used to iterate over ranges of integers. One can create ranges of integers using the `range` type. Here is some code that prints a sequence of integers:

```
1 for i in range(10):
2     print(str(i), end=" ")
```

```
$ python3 range.py
0 1 2 3 4 5 6 7 8 9
```

We can also "nest" loops by putting a loop within a loop. This is useful in many situations, but it can get a bit unwieldy after the third level of nesting. (This is why we will use functions next lecture!)

Here is some code that prints a matrix using a "nested" `for` loop:

```
1 for i in range(5):
2     print(str(i) + "\t", end="")
3     for j in range(5):
4         print(j, end="\t")
5     print()
```

```
$ python3 matrix.py
0: 0 1 2 3 4
1: 0 1 2 3 4
2: 0 1 2 3 4
3: 0 1 2 3 4
4: 0 1 2 3 4
```

Group Work

Write Python code to compute the following tasks using `while` loops, then do the same with `for` loops:

- print all of the integers in the range $[i, j)$
- print all of the integers in the range $[i, j)$ that are even
- print all of the integers in the range $[i, j)$ that are divisible by d
- print all of the integers in the range $[i, j)$, backwards
- print the first n the integers in the range $[i, j)$ that are divisible by d

Write Python code to compute the following task using whatever method you'd like:

- print a box with a perimeter formed by the '$\star$' character. Each side should be made of n individual '$\star$' characters, and the center should be hollow (formed by the blank ' ' character).