CptS 111, Spring 2023
Lect. #8, Feb. 6, 2023
Class Notes

Today's Agenda:

1. Conditionals
2. `if` statements
3. Relational (comparison) operators
4. `if` - `else` statements
5. `if` - `elif` - `else` statements

Ch. 4

Conditionals

A basic and important aspect of programming is the ability to code test expressions (conditions) that evaluate to the Boolean variables `True` or `False`. These *conditionals* allow us to execute different commands depending on the outcome of the test expression.

1. `if` Statements

The simplest conditional is an `if` statement.

`if` statement template:

```
if <test expression>:
    <conditional body>
```

If the test expression evaluates to `True`, then the statements in the body are executed; if it evaluates to `False`, they aren't executed. The body of the conditional must be indented as shown.

Note that all variables, names, functions, literals, classes, and so forth in Python are `True` *except for four.* These four are `False`, `0`, `None`, and anything empty, e.g., an empty string or list. Thus, `True`, `1`, `Hello`, `3.141592`, and so on all evaluate to `True`.

Consider the simplest examples of conditionals--single variable tests:

In [1]:
```python
# We can use a single variable as our test expression
# We can even use the Boolean variables True and False

if True:
    print('This is a true statement.')
```

```
This is a true statement.
```

In [2]:
```python
# Any number that's non-zero evaluates to True

if -1:
    print('This is a true statement.')
```

```
This is a true statement.
```

In [3]:
```python
# Any string that isn't empty evaluates to True

if 'Henry':
    print('This is a true statement.')
```

```
This is a true statement.
```

In [4]:
```python
# Four objects evaluate to False; zero is one of them

if 0:
    print('This is a false statement.')
```

In [5]:
```python
# All empty objects evaluate to False, including an empty string

if '':
    print('This is a false statement.')
```

In [6]:
```python
# None also evaluates to False

if None:
    print('This is a false statement.')
```

Let's next move on to a more complex text expression.

In [7]:
```python
# This is a very useful test!
# The modulo of an even number is 0!

test_num = 6
if (test_num % 2) == 0:          # Parentheses aren't needed
    print('test_num is even!')
```

```
test_num is even!
```

The last example (and variants of it) can actually be quite useful. If we replace `2` by any other number, we can determine whether `test_num` is divisible by that number.

In [8]:
```python
# Check to see whether 7 is a factor of a number

number = 3456789
if (number % 7)  == 0:
    print(f'{number} is divisible by 7.')
```

```
3456789 is divisible by 7.
```

2. Comparison (Relational) Operators

It's useful to be able to use comparison/relational operators in test expressions to compare two operands as in the last two examples, i.e., we used `==` . There are six different comparison/relational operators:

x < y	Is x less than y?
x <= y	Is x less than or equal to y?
x == y	Is x equal to y?
x >= y	Is x greater than or equal to y?
x > y	Is x greater than y?
x != y	Is x not equal to y?

It's best not to use the equality operator (==) with floats because of float precision, i.e., we don't always know the actual value of the float. Note that `==` has to be used to determine whether two things are equal because `=` is the assignment operator.

Let's consider some examples using comparison operators.

In [9]:
```python
# Not equal to

x = 5
y = 10
if x != y:
    print('x and y are not equal.')
```

```
x and y are not equal.
```

In [10]:
```python
# We can use operations in conditionals!

if (x + 5) >= y:
    print("It's not easy to tell how x and y are related.")
```

```
It's not easy to tell how x and y are related.
```

In [11]:
```python
# Another example of an operation

if (5 * x) > y:
    print('5x is greater than y.')
```

```
5x is greater than y.
```

In [12]:
```python
# Less than or equal to

if x <= y:
    print('x is less than or equal to y.')
```

```
x is less than or equal to y.
```

There are instances when a series of `if` statements is the correct way to write a program. Consider the following:

In [13]:
```python
# Use of multiple if statements

age = 58
member = True        # Note that True and False aren't strings so
coupon = False       #  we don't use quotes with them

discount = 0         # Initialize discount to 0
                     # Here discount is an accumulator
if age >= 55:
    discount += 5
if member:
    discount += 5
if coupon:
    discount += 5

print(f'Your discount is {discount}%.')
```

```
Your discount is 10%.
```

For the example above, each `if` statement was required in order to obtain the correct discount. **However, if we only want to execute one body of statements depending on the outcome of a test expression, we don't use a series of `if` statements, or it's considered to be bad coding.**

3. `if-else` Statements

Consider the following example:

In [14]:
```python
# Example of bad coding

age = int(input('Enter your age: '))
if age >= 18:
    print("You're eligible to vote.")
if age < 18:
    print("You're still too young to vote.")
```

```
Enter your age: 28
You're eligible to vote.
```

For this example, it doesn't make sense to execute the second `if` statement, i.e., the second test expression, if the first one evaluates to `True` because the person is either 18 or older or else younger than 18. Instead we should use an `if-else` statement.

`if-else` statement template

```
if <test expression>:
    <body>
else:
    <body>
```

Using this format, the previous example would simply be:

In [15]:
```python
# Example of correct coding using if-else

age = int(input('Enter your age: '))
if age >= 18:
    print("You're eligible to vote.")
else:
    print("You're still too young to vote.")
```

```
Enter your age: 28
You're eligible to vote.
```

Let's consider a few more examples:

In [16]:
```python
# Another if-else example

mile_pace = float(input('Enter their mile time: '))
if mile_pace < 4:
    print('They broke a 4-minute mile.')
else:
    print('They did not break a 4-minute mile.')
```

```
Enter their mile time: 5.67
They did not break a 4-minute mile.
```

In [17]:
```python
# An if-else example with variables

num_cookies = 24
num_children = 25
if num_cookies >= num_children:
    print('We have enough cookies!')
else:
    print('We need to buy more cookies.')
```

```
We need to buy more cookies.
```

4. `if-elif-else` Statements

When we want to execute only one body of statements depending on the outcome of mutiple test expressions, we use an `if-elif-else` construct (you need to use this in zyLab_PA3).

`if-elif-else` statement template

```
if <test expression 1>:
    <body 1>
elif <test expression 2>:
    <body 2>
elif <test expression 3>:
    <body 3>
.
.
.
else:
    <body N>
```

`elif` is shorthand for `else if` . You can have as many `elif` statements as necessary, and then you close with an `else` statement. For example, consider the following:

In [18]:
```python
# Example of if-elif-else

degrees = float(input('Enter current temperature: '))
if degrees <= 20:
    print('Bundle up in lots of clothing!')
elif degrees <= 32:
    print('Wear a warm coat, a scarf, a hat, and gloves.')
elif degrees <= 45:
    print('A warm coat and gloves should be fine.')
elif degrees <= 55:
    print('A jacket should be plenty.')
elif degrees <= 65:
    print('You may need a sweater or hoodie.')
else:
    print('Time to haul out the shorts!')
```

```
Enter current temperature: 43
A warm coat and gloves should be fine.
```

In [19]:
```python
# And one more if-elif-else example using !=

num = int(input('Enter an integer: '))
if num < 0:
    print('We only want to test positive integers.')
elif (num % 3) != 0:
    print('Your integer is not divisible by 3.')
else:
    print('Your integer is divisible by 3.')
```

```
Enter an integer: 42
Your integer is divisible by 3.
```

In [20]:
```python
# We can use any number we want; we can also use == rather than !=

num = int(input('Enter an integer: '))
if (num % 7) == 0:
    print('Your number is divisible by 7.')
else:
    print('Your number is not divisible by 7.')
```

```
Enter an integer: 42
Your number is divisible by 7.
```