Test 01 is this week!

- review page for Test 01:
  - https://homes.sice.indiana.edu/classes/fall2018/csci/a201-mitja/2018/tests/a201-test01-review.html
Review: Indentation and Code Blocks

Correct:

```python
if password == "secret":
    print "Access Granted"
else:
    print "Access Denied"
```

Incorrect:

```python
if password == "secret":
    print "Access Granted"
else:
    print "Access Denied"
```

Indenting, i.e., using spaces, creates code blocks, units of code, which have a meaning in the program flow.
Review / Lecture Task: Odd or Even

• Ask the user for a number, then print out if it’s odd, even, or not a whole number.
• Experiment with Interactive Mode to see how you can tell!

• Useful Code:

```python
if Boolean Condition:
    # Whatever we want to happen if this condition is true
elif Other Boolean Condition:
    # Whatever we want to happen if this condition is true
else:
    # This will happen only if all of the above conditions are false!

== # test for equality
% # modulus operator
```

```python
# modify this way:
# 1. ask the user for a NUMBER
# 2. tell them if it's ODD, EVEN, or NOT a whole number
# == tests for equality
# % mod or modulus operator
# one more hint... float() is useful!

myPassword = input("enter your passw: ")

if myPassword == "secret":
    print ("access granted")
elif myPassword == "oldsecret":
    print ("sorry, it's the old one")
else:
    print ("sorry wrong password")
```
Review: What Can We Compare?

- what makes sense to compare:
  - Integer to Integer
  - Float to Float
  - Integer to Float
  - String to String (later we’ll see how)

- what does not make sense to compare:
  - String to Integer
  - String to Float

  (...unless you know what you're doing?!? when would it work???)
True or What?!? Treating Values as Conditions

- Any value can be interpreted as True or False, as a Boolean condition.
  - Any empty (None) or zero value is False:
    - So for example, 0, '', and None are False
  - Any other value is True:
    - So for example, -10, 2.5, "a string" are True

- Try it yourself... type the following code and run it →

```python
if 0:
    print("0 is True")
else:
    print("0 is False")

if ":
    print("" is True")
else:
    print("" is False")

if None:
    print("None is True")
else:
    print("None is False")

if -10:
    print("-10 is True")
else:
    print("-10 is False")

if 2.5:
    print("2.5 is True")
else:
    print("2.5 is False")

if "a string":
    print(""a string" is True")
else:
    print(""a string" is False")
```

- This also works with variables. for example if myAge is a variable:

  ```python
  if myAge:
    myAge is treated as condition
    it's going to be True when myAge is not 0, nor ", nor None
    it's going to be False when myAge is 0, or ", or None
  ```
Review: Logic: Compound Conditions

- Sometimes, we’d like to test multiple conditions at once. for example:
  - System login: username AND password must both be correct
  - System login: if the user does NOT enter a username, we’ll need to ask again
review - another logic operator: NOT

- **not** is a logical operator that reverses the value of any Boolean Condition

```python
username = ""

if not username:
    username = input("Enter Username: ")
```

- (remember, "" evaluates to False...)
Review: AND

- **and** is a logical operator that connects two Boolean Conditions. It’s true if they are BOTH true, and false otherwise.

- for example:

```python
myNumber = float(input("enter a number: "))

if (myNumber % 2 == 0) and (myNumber < 100):
    print ("it was both EVEN and smaller than 100")
    print ("the result was", (myNumber % 2 == 0) and (myNumber < 100))
```
Review: OR

- or is a logical operator that connects two Boolean Conditions. It’s true if either of them is true OR if they both are! for example:

```python
myNumber = float(input("enter a number: "))

if (myNumber % 2 == 0) and (myNumber < 100):
    print ("it was both EVEN and smaller than 100")
    print ("the result was", (myNumber % 2 == 0) and (myNumber < 100))
elif (myNumber % 2 == 1) or (myNumber > 100):
    print ("it was either ODD or larger than 100, or both.")
```

- ... however, is this the way we use OR in English?
An immutable data type: *Tuples*

- a **tuple** is an *immutable sequence of values of any type*
  - (compare: a string is an immutable sequence of characters)

- for example: use a *tuple of integers* for a high score list

- the elements of a *tuple* don't need to all be of the same type
Tuple Basics

- Creating an Empty Tuple
  ```python
  >>> inventory = ()
  >>> if not inventory:
      print( "You are empty-handed. ")
  
  You are empty-handed.
  ```

- Treating a Tuple as a Condition
  ```python
  >>> inventory = ("sword", "armor", "shield", "healing potion")
  >>> inventory
  ('sword', 'armor', 'shield', 'healing potion')
  >>>
  >>>
  >>> if not inventory:
      print( "You got nothing. ")
  ```

- Creating a Tuple with Elements
Tuple Basics (continued)

```python
>>> print( "The tuple inventory is:\n" + str(inventory) )
The tuple inventory is:
('sword', 'armor', 'shield', 'healing potion')
>>> for item in inventory:
    print(item)
sword
armor
shield
healing potion
```

- Printing a tuple
- Looping through a tuple's elements...
  what is that?!?

A new Python statement that we haven't seen before: the **for** loop!

(no worries, we don't have to know the **for** loop for Test 01!)
Using Tuples

• Tuples are a kind of **sequence** (like strings) so you can:

  • Get a tuple's length with `len()`
  • Iterate through elements with `for` loop
  • Test for element membership with `in`
  • Index, slice, and concatenate

• *let's look at each one of these, one at a time...*
Using `len()` and `in` with Tuples

- **The `len()` function with tuples**
  - just as with strings, `len()` returns the number of elements in a tuple:

```
>>> print("You have " + str(len(inventory)) + " items.")
You have 4 items.
```

- **The `in` operator with tuples**
  - just as with strings, `in` tests for element membership in a tuple:

```
>>> if "healing potion" in inventory:
    print("you're healthy again!")

you're healthy again!
```
Indexing Tuples

- Each element has a corresponding position number.
- Each string is a single element in the tuple.
Slicing Tuples

- Slicing positions defined between elements
- Tuple slicing works much like string slicing.
Tuple Immutability

>>> inventory = ("sword", "armor", "shiel", "healing potion")
>>> inventory
('sword', 'armor', 'shield', 'healing potion')

>>> inventory[0] = "foil"
Traceback (most recent call last):
  File "<pyshell#68>", line 1, in <module>
    inventory[0] = "foil"
TypeError: 'tuple' object does not support item assignment

• just like strings, tuples are immutable:
  you can't change just one element in a tuple

• but you can create a new tuple from an existing one
Concatenating Tuples

```python
>>> inventory = ("sword", "armor", "shield", "healing potion")
>>> chest = ("gold", "gems")
>>> inventory = inventory + chest
>>> inventory
('sword', 'armor', 'shield', 'healing potion', 'gold', 'gems')
```
Summary about Sequences

• An ordered list of elements is called what?
  • A sequence

• To move through a sequence, in order, is called what?
  • Iterate

• What would `range(20, 10, -2)` return?
  • `[20, 18, 16, 14, 12]`

• What would `len(range(20, 10, -2))` return?
  • 5
Summary about Sequences (continued)

- If I use the `in` operator to test for element membership in a tuple, what does it return if the element is there?
  - True

- What is the name of the technique used to access a specific element of a sequence?
  - Indexing

- Match the following pairs of words:
  - mutable \(\leftrightarrow\) unchangeable
  - immutable \(\leftrightarrow\) changeable

- Strings are immutable sequences, true or false?
  - True

- Constants are values that are meant to change, true or false?
  - False
Summary about Slicing and Tuples (continued)

• String concatenation adds onto an existing string, true or false?
  • False, it creates brand-new strings

• What does None evaluate to when treated as a condition?
  • False

• Slicing creates a copy of a discontinuous collection of elements from a sequence, true or false?
  • False, it only copies a continuous segment of elements from a sequence

• A tuple is an immutable sequence of elements of what variable type?
  • Any!

• The concatenation operator, +, works with tuples just like with strings, true or false?
  • True