In-class worksheet 15

Mar 11, 2019

Introduction to `if` statements

The `if` statement allows you to execute pieces of code only if a given **logical statement** is True. Logical statements are any piece of code which yields a True or False value (typically, used with ==, `is`, >, <, etc).

`if` statements follow the basic format

```
if <logical statement is True>:
    do this code
```

Examples:

In [1]:
```
# This if statement evaluates as True
a = 5
if a < 10:
    print("a is less than 10!")
```

```
a is less than 10!
```

In [2]:
```
# This if statement evaluates as False.
# Note that there is NO output!
a = 5
if a > 10:
    print(a, "is greater than 10")
```

`if` statements can be used in conjunction with `else` statements to create `if/else` statements. These statements execute code inside the `if` if the logical statement is True, and if the statement is False, code inside the `else` block is executed.

`if/else` statements follow the basic format

```
if <logical statement is True>:
    do this code
else:
    do this code instead
```

Examples:

In [3]:
```
# Either code inside the "if" or code inside the
# "else" gets executed - not both!
a = 5
if a > 10:
    print(a, "is greater than 10")
else:
    print(a, "is less than or equal to 10")
```

```
5 is less than or equal to 10
```

There are also constructs known in python as `elif` ("else if"), which allow you to test multiple conditions.

`if/elif/ese` statements follow the basic format

```
if <logical statement is True>:
    do this code
elif <some other logical statement is True>:
    do this code instead
else:
    do this code if nothing above was true.
```

You can have as many `elif` statements, following an initial `if` statement, as you want. It is not strictly necessary to end with `else`, but typically you will want to do so. In a given `if/elif.../else` construct, only **one** of the conditions will run. Should an `if` or `elif` evaluate to True, python will simply run that code and ignore the rest.

Examples:

In [4]:
```
a = 6
b = 22
if a > b:
    print(a, "is greater than", b)
elif a < b:
    print(a, "is less than", b)
else:
    print(a, "is equal to", b)
```

```
6 is less than 22
```

Note that multiple `if` statements in a row are entirely distinct. If you have an `if` statement followed by another `if` statement, they can both be run. Further, `elif` and `else` statements are connected only to the most recent `if` statement preceding them.

In [5]:
```
s = "elephant"
if len(s) < 50:
    print(s, "is shorter than 50 characters")
if len(s) < 30:
    print(s, "is shorter than 30 characters")
else: # this goes with `len(s) < 30`, NOT with `len(s) < 50`
    print(s, "is longer than 30 characters")
```

```
elephant is shorter than 50 characters
elephant is shorter than 30 characters
```

Problems

Problem 1:

a) Define a numeric variable (either float or integer is ok). Use an `if`/`else` statement to determine if the number is greater than zero. Your code should print a sentence indicating if the number is greater than zero or not.

b) Modify your `if`/`else` statement to write an `if`/`elif`/`else` statement to determine if the number is greater than, less than, or equal to zero. Again, print a sentence indicating the number's value relative to 0.

In [6]:
```
# if/else statement
b = 926
if b > 0:
    print(b, "is greater than 0.")
else:
    print(b, "is not greater than 0.")

# if/elif/else statement
b = -15
if b > 0:
    print(b, "is greater than 0.")
elif b < 0:
    print(b, "is less than 0.")
else:
    print(b, "is equal to 0.")
```

```
926 is greater than 0.
-15 is less than 0.
```

Problem 2:

a) Write an if/else statement to test the single condition, whether the length of the list `mylist` (defined below) is less than or equal to 10. If this condition is *true*, use indexing to create a new list called `newlist` that contains the first three numbers `mylist`. If this condition is *false*, use indexing to create a new list called `newlist` that contains the first 7 numbers in the list.

Once `newlist` is defined, determine and print its sum. (Hint: use the function `sum()`.)

b) Use an if/else statement to determine if the sum is even or odd. Print your result. (Hint: use the modulus operator `%` to test even vs. odd.)

In [7]:
```
# Use this list to solve Problem 2:
mylist = [19, 3, 2, 88, 56, 57, 11, 19, 9, 95]

# Create newlist based on length of mylist
if len(mylist) <= 10:
    newlist = mylist[:3]
else:
    newlist = mylist[:7]

# Determine the sum of newlist
result = sum(newlist)
print("The sum of newlist is", result)

# Determine if result is even or odd
if result % 2 == 0:
    print("The sum is even.")
else:
    print("The sum is odd.")
```

```
The sum of newlist is 24
The sum is even.
```

Problem 3: Write an if/elif.../else construct to determine the type of the variable `a`, defined below.

Your code should evaluate if the variable is one of the following types: integer (int), float (float), string (str), or list (list). Your code should print a sentence stating the variable type (e.g. "89.44 is a float").

In [8]:
```
# Use this variable to solve Problem 3:
a = 89.44

if type(a) is int:
    print(a, "is an integer.")
elif type(a) is float:
    print(a, "is a float.")
elif type(a) is list:
    print(a, "is a list.")
elif type(a) is str:
    print(a, "is a list.")
```

```
89.44 is a float.
```

Introduction to `for` loops

`for` loops are used in two main circumstances:

1. to perform a certain operation on each item in a list, string, dictionary, etc.
2. to perform a certain operation a specific number of times

They follow the basic format

```
for item in container:
    do this command
    do that command
```

The `item` is known as a "loop variable". At each iteration of the loop, the variable `item` takes on a specific value corresponding to that iteration. This variable will change at each new iteration of the loop.

Examples of looping over lists, strings, dictionaries:

In [9]:
```
# Loop over a list
mylist = [1, 2, 3, 4]
for i in mylist:
    print(i)
```

```
1
2
3
4
```

In [10]:
```
# Loop over a string
for letter in "abcdefg":
    print(letter)
```

```
a
b
c
d
e
f
g
```

In [11]:
```
# Loop over a dictionary (loops over the keys!)
d = {"a":1, "b":2, "c":3, "d":4}
for key in d:
    print(key)    # prints the key
    print(d[key]) # prints the key's value
    print()       # using print on its own will print a blank line
```

```
b
2

a
1

c
3

d
4
```

Oftentimes, counter variables are used in loops to keep track of which iteration you're on.

In [12]:
```
i = 0 # Define counter variable
for x in [3, 6, 9, 12]:
    print(x)
    i += 1 # Increment the variable i by 1 with the += operator
    print("We just finished iteration number", i)
```

```
3
We just finished iteration number 1
6
We just finished iteration number 2
9
We just finished iteration number 3
12
We just finished iteration number 4
```

To perform an action a specific number of times, the `range()` statement is commonly used:

In [13]:
```
# `range(10)` corresponds to all numbers from 0 to 9:
for i in range(10):
    print(i)
```

```
0
1
2
3
4
5
6
7
8
9
```

In [14]:
```
# `range(2, 13)` corresponds to all numbers from 2 to 12:
for i in range(2, 13):
    print(i)
```

```
2
3
4
5
6
7
8
9
10
11
12
```

In [15]:
```
# `range(4, 20, 2)` corresponds to all numbers from 4 to 19 in steps of 2:
for i in range(4, 20, 2):
    print(i)
```

```
4
6
8
10
12
14
16
18
```

Problems

Problem 1:

a) Write a `for` loop that iterates over the numbers 0-10 (including 10!). On each iteration, print 2 raised to that iteration count. For example, when the loop variable equals 3, your code should print 8. (Hint: In Python, powers are calculated with the ** operator.)

b) Modify your code from part (a) to save each of those power-of-2 values to a list called `powers` (hint: use the `.append()` method). Print the final `powers` list.

In [16]:
```
# Print the powers of 2
for i in range(11):
    print(2**(i))
```

```
1
2
4
8
16
32
64
128
256
512
1024
```

In [17]:
```
# Save powers to a list
powers = [] # make empty list
for i in range(11):
    powers.append(2**i) # add power value to list
print(powers) # print the final list
```

```
[1, 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024]
```

Problem 2: Two variables are defined in the code chunk below: the string `protseq`, which contains a sequence of amino acids, and the dictionary `amino_weights`, which gives the molecular weight for each amino acid. Write a `for` loop to determine the total molecular weight of `protseq`. Print the total weight.

In [18]:
```
# Use these two variables to solve Problem 2:
protseq = "RTAHHCPLKLLAWS"
amino_weights = {'A':89.09, 'R':174.20, 'N':132.12, 'D':133.10, 'C':121.15,
                 'Q':146.15,'E':147.13,'G':75.07,'H':155.16,'I':131.17,
                 'L':131.17,'K':146.19,'M':149.21,'F':165.19,'P':115.13,
                 'S':105.09,'T':119.12,'W':204.23,'Y':181.19,'V':117.15}

weight = 0. # Start with zero weight. The period in 0. is critical,
            # it tells Python that `weight` is a floating-point number.
for aa in protseq:
    weight += amino_weights[aa]
print(weight)
```

```
1867.12
```

Introduction to combining `if` and `for`

The `if` and `for` constructs are frequently combined, for example when we loop over all elements in a list and check whether the elements meet some condition.

Examples:

In [19]:
```
# Evaluate a condition on every element in a list
mylist = [8, 472, -185, 0, -778.2, 23, 90, -0.003]
for i in mylist:
    if i > 0:
        print(i, "is positive")
    elif i < 0:
        print(i, "is negative")
    else:
        print(i, "is 0")
```

```
8 is positive
472 is positive
-185 is negative
0 is 0
-778.2 is negative
23 is positive
90 is positive
-0.003 is negative
```

In [20]:
```
# Find the sum of all the multiples of 3 that are <=30.
result = 0              # this variable will hold our result
for i in range(1, 31):
    if i%3 == 0:        # the % sign is the modulo operator, which produces the
remainder after division
        result += i
print("The sum of all multiples of 3 which are <= 30 is", result)
```

```
The sum of all multiples of 3 which are <= 30 is 165
```

In [21]:
```
# test whether a sequence is RNA or DNA by seeing if it contains the letter U
seqs=['AUUGAC', 'AGACT', 'CGATAGCA', 'UCCAGAC', 'UGGACU', 'TAGCAGA']
for seq in seqs:
    if 'U' in seq and 'T' not in seq:
        print(seq, "is probably RNA")
    elif 'T' in seq and 'U' not in seq:
        print(seq, "is probably DNA")
    elif 'T' in seq and 'U' in seq:
        print(seq, "has both T and U. I don't know what it is!")
    elif 'T' not in seq and 'U' not in seq:
        print(seq, "has neither T nor U. I don't know what it is!")
    else:
        print("Error: This line should never be executed.")
```

```
AUUGAC is probably RNA
AGACT is probably DNA
CGATAGCA is probably DNA
UCCAGAC is probably RNA
UGGACU is probably RNA
TAGCAGA is probably DNA
```

Problems

Problem 1: Loop over the dictionary d, defined below. For each key, determine if it contains the letter "o". If this condition is true, print the *value* from d associated with that key. Otherwise, print the statement "Sorry, no 'o' in the key XXX", where XXX is the name of the key.

In [22]:
```
# Use this dictionary to solve Problem 1:
d = {"frog":"amphibian",
     "crocodile":"reptile",
     "osprey":"bird",
     "platypus":"mammal",
     "squid":"mollusk",
     "spider":"arachnid",
     "jellyfish":"cnidarian",
     "clownfish":"fish"}

for item in d:
    if "o" in item:
        print(d[item])
    else:
        print("Sorry, no 'o' in the key", item)
```

```
reptile
bird
Sorry, no 'o' in the key squid
Sorry, no 'o' in the key platypus
Sorry, no 'o' in the key jellyfish
Sorry, no 'o' in the key spider
fish
amphibian
```

Problem 2: Two variables are defined in the code chunk below: the string `protseq_ambig`, which contains a sequence of amino acids, and the dictionary `amino_weights`, which gives the molecular weight for each amino acid. Unlike the previous exercise on molecular weight, this time the amino acid sequence contains ambiguous amino-acid characters, including X, B, and Z, which are *not* in the amino_weights dictionary. Write a `for` loop to determine the total molecular weight of `protseq_ambig`, ignoring all amino acids that are ambiguous. Print the total weight.

In [23]:
```
# Use these two variables to solve Problem 2:
protseq_ambig = "TAGHHABAAXARBYEDEEZKMBPXQ"
amino_weights = {'A':89.09, 'R':174.20, 'N':132.12, 'D':133.10, 'C':121.15,
                 'Q':146.15,'E':147.13,'G':75.07,'H':155.16,'I':131.17,
                 'L':131.17,'K':146.19,'M':149.21,'F':165.19,'P':115.13,
                 'S':105.09,'T':119.12,'W':204.23,'Y':181.19,'V':117.15}


weight = 0.
for aa in protseq_ambig:
    if aa in amino_weights:
        weight += amino_weights[aa]
print(weight)
```

```
2436.5200000000004
```