Intro to Python (Part II)

Kseniya Usovich

Introduction to Python

Table of Contents

b - While Loops

3 - Functions

4 - Libraries

Dependencies:

In [ ]:
```
import numpy as np
import pandas as pd
import math
```

Conditionals

Why do we care about conditionals?

- Use conditionals statements to control when you want a block of code to execute.
  - if x < 2: print("Hello")
- Use inside loops
- Output is a boolean data type.

In [ ]:
```
# EXAMPLE

0 < -1
```

In [ ]:
```
# EXAMPLE

abs(-2) < abs(2)
```

Now, let's check the type of each of the examples above. We will using the **type** function.

In [ ]:
```
# EXAMPLE
type(0< -1)
```

In [ ]:
```
# EXAMPLE
type(abs(-2) < abs(2))
```

Operators

Here is a list with more **operators**!

Operator	Meaning
<	less than
<=	lass than or equal to
>	greater than
>=	greater than or equal to
!=	not equal to
==	equal to

Let's try some excercises:

Exercise 1 Write "4 equal to 'four'" in code using conditionals.

In [ ]:
```
# YOUR CODE
```

Exercise 2 Write "2 equal to 1 + 1" in code using conditionals.

In [ ]:
```
# YOUR CODE
```

Exercise 3 Write "0 equal to False'" in code using conditionals.

In [ ]:
```
# YOUR CODE
```

Exercise 4 How would you express 30 in the interval 28 to 50, exclusive?

In [ ]:
```
# YOUR CODE
```

A conditional statement

A conditional statement has the following form:

```
if <condition>:
    <do_something>
elif <condition>:
    <do_something>
else:
     <do_something>
```

The **condition** called **expressions** and **do_something** are called **suite**. (Don' worry about terminology for now.)

In [ ]:
```
# EXAMPLE

if 5<6:
    print("True. 5 is less than 6!")
else:
    print("False. What is math?")
```

How do conditional statements work?

- First, it evaluated the header expression(which is the `if` statement). If the statement is True, then it executes the first suite otherwise, it checks the condition of the `elif`, and so on.
- It will only execute the suite (do_something) of the `else` clause, if the conditions of the `if` and `elif` are false (not met).

Some notes about conditional statements:

- You can have as many `elif` expressions as you want, but you can only have one `if` and one `else` statement.
- `else` always comes at the end.
- You do not always need and `else`.
- When you execute a conditional statement, you must ALWAYS consider the order of you statements.

In [ ]:
```
# EXAMPLE 1

if len("Karla") == 5:
    print("Hello, 5!")
else:
    print("Hello,6!")
```

In [ ]:
```
# EXAMPLE 2

fizzbuzz =21
if fizzbuzz % 3 == 0 and fizzbuzz % 5 == 0:
    print("fizzbuzz")
elif fizzbuzz % 3 == 0:
    print("fizz")
elif fizzbuzz % 5 == 0:
    print("buzz")
```

In [ ]:
```
# EXERCISE 1
```

```
# if the strings is equal to NERDS, print "Cal NERDS", if n
ot print "Geek".
string1 = "NERDS"

if ...:
    print("Cal NERDS")
else:
    print("...")
```

In [ ]:
```
# Once you completed the code above, try it with a new stri
ng where the string is NOT "NERDS".
# We have suggeste string2.

string2 = "NOT A NERD"
# COPY CODE FROM ABOVE BUT CHANGE STRING1 to STRING2

...
```

Loops

What are loops useful for?

- Sometimes we want to perform some operation on a list, or tuple, but doing it on each element individually can be very tedious!
- A loop goes through a collection of data types and **executes** some state statement **for each value** in that collection of data.
- Examples of data types are: list, array, characters string, tuple, dictionary, and other collections

---

"For" Loops

Syntax:

for var **in** iterable:

```
    statement
    statement
```

else:

```
statement
```

- var - it takes items from iterable one by one.
- iterable - it's a collection of objects (list tuple)
- indentation - loop body MUST be indented
- loop body - first two statements
- else clause - last statement.

In [ ]:
```
# EXAMPLE

# square each value in the tuple

for x in (1,2,3):
    print(x**2)
```

In [ ]:
```
# EXAMPLE

# print each item in the list.
# recall that the elements in the list do not have to have
the same type

for i in [1, 2.0, "k"]:
    print(i)

i
```

Create a for-loop that multiplies every number in the list by 3 (and outputs it on every step).

In [ ]:
```
# EXERCISE

numbers = [1, 5, 7, 0, 10, 4, 7]

for i in ...:
    ...
```

In [ ]:
```
# EXAMPLE

# print each animal in the animals list

animals = ["dog", "cat", "bird", "fish", "bear"]
new_animal = []

for animal in animals:
    print(new_animal)
```

```
    new_animal += [animal]
    print(new_animal)
```

In [ ]:
```
# EXAMPLE


# calculate the area for circles with different radius
# you can save the list in variable before iterating over i
t
radiuses = [1, 3, 5, 7, 9]

for radius in radiuses:
    area = math.pi*radius**2
    new_area = area +50
    print(new_area)

area
```

In [ ]:
```
# EXAMPLE

# change each letter in the random_sentence from lower to u
pper case
random_sentence = "A very long word"

for letter in random_sentence:
    print(letter.upper())
```

As we mentioned before, you can loop through any iterable data type or data sctructure. That means that we can even loop through dictionaries.

In [ ]:
```
# EXAMPLE

cities = {'New York': 1, 'Los Angeles': 2, 'Miami': 3, 'Chi
cago': 4}

for key in cities:
    print(key)
```

In the cell below, try looping through the values instead of keys.

In [ ]:
```
# EXERCISE

for value in cities:
    print(...)
```

"While" Loops

Syntax:

while condition:

```
    statement
    statement
```

else:

```
    statement
```

- condition - any expression that evaluates to true or false (so, our conditionals!)
- loop body - first and second statements. It is excludes as long as the condition is true.
- else clause - it is executed if the condition becomes false
- indentation - loop body must be indented.

In [ ]:
```
# EXAMPLE

# Iterate until x becomes 0
x = 10
while x:
    print(x)
    x = x- 1
```

In [ ]:
```
# EXAMPLE

# iterates until the string is empty
x = 'I can code!'
while x:
    print(x)
    x = x[1:]
```

In [ ]:
```
# EXAMPLE

x = 10
while x:
    print(x)
    x = x - 1
else:
    print('Done with my countdown!')
```

In [ ]:
```
# EXERCISE

# Create a while loop that skips even numbers and squares i
t

x = 10
while ...:
    ...
    ...
    if x % 2 != 0 :
        ...
```

Functions

Built-in Functions

Python does not have a lot of built-in (aka available by default) functions and methods. But it does have some. We will go over a few useful functions and methods that will be good for you to know.

If something is a function, it will appear in green color.

In [ ]:
```
# EXAMPLE

max(2, 5, 9, 7)
```

In [ ]:
```
# EXAMPLE

min(3, 7)
```

How will this function work with textual information (aka strings)? Pause for a moment to answer this question before you run this cell.

In [ ]:
```
# EXAMPLE
```

```
max("dog", "Dog")
```

Another built-in function you will most likely use in your coding a lot, is **len( )**. It is used to learn about the length of different objects.

In [ ]:
```
# EXAMPLE

len("dog")
```

In [ ]:
```
# EXERCISE

# what do you think this cell will output?


len("dog ")
```

When used with lists, it counts only the "outter" objects. Meaning, if there are lists or arrays inside of your list, no matter however many objects each of them has, the len( ) will count the whole list (array, tuple, dictionary) as a single object.

In [ ]:
```
# EXAMPLE


len([1,3,4])
```

In [ ]:
```
random = [[1,2,3,4],[4,5,6]]

# Before running this cell, can you guess the output
# it will give you?

len(random)
```

In the example below, we will use a built-in method. There's no big difference between the two except for the syntax.

Functions (not only the built-in ones) are usually used in the form of

function(argument 1, argument n)

Unlike functions, methods are usually added at the end of the variable.

variable.method()

In [ ]:
```
# EXAMPLE
```

```
hello = "Hello World"

print(hello.upper())
print(hello)
```

Have you noticed that the variable **hello** didn't change? Can you think why is that?

In [ ]:
```
# EXAMPLE


hello = hello.lower()
hello
```

In the cell below, create a variable **phrase** which tells us what school you go to. Then change all the letters to upper-cased letters.

In [ ]:
```
# EXERCISE


phrase = "I go to ..."
...

print(phrase)
```

Defining Fucntions

A function is a block code that you can reuse to perform a specific action. In the next cell, we will be defining a simple one-line function that takes one argument.

In [ ]:
```
# EXAMPLE

def square(x):
    return x**2


a = square(3)
a
```

Your functions can perform multiple calculations and use more than one argument. You can also incorporate for-loops, conditionals, and other functions inside of your function.

Note: we won't be going over the nested functions (functions with other functions inside, but feel free to search for them)

feel free to search for them).

In the cell below, we are going to define a function that will multiply the two numbers you give it only if they are not equal to each other.

In [ ]:
```
# EXAMPLE

def mult_not_eq(x, y):
    # if the first number is not equal to the second
    print(x, y)
    if x!=y:
    # I will multiply them
        return x*y
    else:
        print("Use the square function instead")


mult_not_eq(2, 2)
```

Now it's time for you to define a function. Create a function that will add up two numbers, but only if they are not equals.