**Introduction to Phyton #1  
Python Basics**

**The Python Interface**

In the Python script on the right, you can type Python code to solve the exercises. If you hit *Run Code* or *Submit Answer*, your python script (script.py) is executed and the output is shown in the IPython Shell. *Submit Answer* checks whether your submission is correct and gives you feedback.

You can hit *Run Code* and *Submit Answer* as often as you want. If you're stuck, you can click *Get Hint*, and ultimately *Get Solution*.

You can also use the IPython Shell interactively by simply typing commands and hitting Enter. When you work in the shell directly, your code will not be checked for correctness so it is a great way to experiment.

**Instructions**

**100 XP**

* Experiment in the IPython Shell; type 5 / 8, for example.
* Add another line of code to the Python script on the top-right (not in the Shell): print(7 + 10).
* Hit *Submit Answer* to execute the Python script and receive feedback.

[**Take Hint (-30 XP)**](javascript:void(0))

Script.py  
1 # Example, do not modify!

2 print(5 / 8)

3 print(7 + 10)

4

5 # Put code below here

6 print(7 + 10)  
  
Run Code  
# Example, do not modify!

print(5 / 8)

print(7 + 10)

# Put code below here

print(7 + 10)

**When to use Python?**

Python is a pretty versatile language. For which applications can you use Python?

**Instructions**

**50 XP**

**Possible Answers**

* 

You want to do some quick calculations.

* 

For your new business, you want to develop a database-driven website.

* 

Your boss asks you to clean and analyze the results of the latest satisfaction survey.

* 

All of the above.

# Any comments?

Something that Filip didn't mention in his videos is that you can add **comments** to your Python scripts. Comments are important to make sure that you and others can understand what your code is about.

To add comments to your Python script, you can use the # tag. These comments are not run as Python code, so they will not influence your result. As an example, take the comment on the right, # Division; it is completely ignored during execution.

##### Instructions

**100 XP**

Above the print(7 + 10), add the comment # Addition.

Script.py  
1 # Division

2 print(5 / 8)

3

4 # Addition

5 print(7 + 10)

IPythonShell  
# Division

print(5 / 8)

# Addition

print(7 + 10)

**Python as a calculator**

Python is perfectly suited to do basic calculations. Apart from addition, subtraction, multiplication and division, there is also support for more advanced operations such as:

* Exponentiation: \*\*. This operator raises the number to its left to the power of the number to its right. For example 4\*\*2 will give 16.
* Modulo: %. This operator returns the remainder of the division of the number to the left by the number on its right. For example 18 % 7 equals 4.

The code in the script on the right gives some examples.

**Instructions**

**100 XP**

Suppose you have $100, which you can invest with a 10% return each year. After one year, it's 100×1.1=110100×1.1=110dollars, and after two years it's 100×1.1×1.1=121100×1.1×1.1=121. Add code on the right to calculate how much money you end up with after 7 years.

Script.py

01 # Addition, subtraction

02 print(5 + 5)

03 print(5 - 5)

04

05 # Multiplication, division, modulo, and exponentiation

06 print(3 \* 5)

07 print(10 / 2)

08 print(18 % 7)

09 print(4 \*\* 2)

10

11 # How much is your $100 worth after 7 years?

12 print(100 \* 1.1 \*\* 7)

# Addition, subtraction

print(5 + 5)

print(5 - 5)

# Multiplication, division, modulo, and exponentiation

print(3 \* 5)

print(10 / 2)

print(18 % 7)

print(4 \*\* 2)

# How much is your $100 worth after 7 years?

print(100 \* 1.1 \*\* 7)

**Variable Assignment**

In Python, a variable allows you to refer to a value with a name. To create a variable use =, like this example:

x = 5

You can now use the name of this variable, x, instead of the actual value, 5.

Remember, = in Python means *assignment*, it doesn't test equality!

**Instructions**

**100 XP**

* Create a variable savings with the value 100.
* Check out this variable by typing print(savings) in the script.

Script.py

1 # Create a variable savings

2 savings = 100

3

4 # Print out savings

5 print(savings)

IPython Shell

# Create a variable savings

savings = 100

# Print out savings

print(savings)

**Calculations with variables**

Remember how you calculated the money you ended up with after 7 years of investing $100? You did something like this:

100 \* 1.1 \*\* 7

Instead of calculating with the actual values, you can use variables instead. The savings variable you've created in the previous exercise represents the $100 you started with. It's up to you to create a new variable to represent 1.1and then redo the calculations!

**Instructions**

**100 XP**

* Create a variable growth\_multiplier, equal to 1.1.
* Create a variable, result, equal to the amount of money you saved after 7 years.
* Print out the value of result.

Script.py

01 # Create a variable savings

02 savings = 100

03

04 # Create a variable growth\_multiplier

05 growth\_multiplier = 1.1

06

07 # Calculate result

08 result = savings \* growth\_multiplier \*\* 7

09

10 # Print out result

11 print (result)

IPython Shell

In [1]: # Create a variable savings

savings = 100

# Create a variable growth\_multiplier

growth\_multiplier = 1.1

# Calculate result

result = savings \* growth\_multiplier \*\* 7

# Print out result

print (result)

194.87171000000012

In [2]:

**Other variable types**

In the previous exercise, you worked with two Python data types:

* int, or integer: a number without a fractional part. savings, with the value 100, is an example of an integer.
* float, or floating point: a number that has both an integer and fractional part, separated by a point. growth\_multiplier, with the value 1.1, is an example of a float.

Next to numerical data types, there are two other very common data types:

* str, or string: a type to represent text. You can use single or double quotes to build a string.
* bool, or boolean: a type to represent logical values. Can only be True or False (the capitalization is important!).

**Instructions**

**100 XP**

* Create a new string, desc, with the value "compound interest".
* Create a new boolean, profitable, with the value True.

Script.py  
1 # Create a variable desc

2 desc = "compound interest"

3

4 # Create a variable profitable

5 profitable = True

IPython Shell  
In [1]: # Create a variable desc

desc = "compound interest"

# Create a variable profitable

profitable = True

In [2]:

**Guess the type**

To find out the type of a value or a variable that refers to that value, you can use the [**type()**](https://docs.python.org/3/library/functions.html#type) function. Suppose you've defined a variable a, but you forgot the type of this variable. To determine the type of a, simply execute:

type(a)

We already went ahead and created three variables: a, b and c. You can use the IPython shell on the right to discover their type. Which of the following options is correct?

**Instructions**

**50 XP**

**Possible Answers**

* 

a is of type int, b is of type str, c is of type bool

* 

a is of type float, b is of type bool, c is of type str

* 

a is of type float, b is of type str, c is of type bool

* 

a is of type int, b is of type bool, c is of type str

In [1]: type(a)

Out[1]: float

In [2]: type(b)

Out[2]: str

In [3]: type(c)

Out[3]: bool

# Operations with other types

Filip mentioned that different types behave differently in Python.

When you sum two strings, for example, you'll get different behavior than when you sum two integers or two booleans.

In the script some variables with different types have already been created. It's up to you to use them.

##### Instructions

**100 XP**

* Calculate the product of savings and growth\_multiplier. Store the result in year1.
* What do you think the resulting type will be? Find out by printing out the type of year1.
* Calculate the sum of desc and desc and store the result in a new variable doubledesc.
* Print out doubledesc. Did you expect this?

Script.py  
01 savings = 100

02 growth\_multiplier = 1.1

03 desc = "compound interest"

04

05 # Assign product of growth\_multiplier and savings to year1

06 year1 = savings \* growth\_multiplier \*\* 1

07

08 # Print the type of year1

09 print(type(year1))

10

11 # Assign sum of desc and desc to doubledesc

12 doubledesc = desc + desc

13

14 # Print out doubledesc

15 print(doubledesc)

IPython Shell  
In [1]: savings = 100

growth\_multiplier = 1.1

desc = "compound interest"

# Assign product of growth\_multiplier and savings to year1

year1 = savings \* growth\_multiplier \*\* 1

# Print the type of year1

print(type(year1))

# Assign sum of desc and desc to doubledesc

doubledesc = desc + desc

# Print out doubledesc

print(doubledesc)

<class 'float'>

compound interestcompound interest

In [2]:

# Type conversion

Using the + operator to paste together two strings can be very useful in building custom messages.

Suppose, for example, that you've calculated the return of your investment and want to summarize the results in a string. Assuming the floats savings and result are defined, you can try something like this:

print("I started with $" + savings + " and now have $" + result + ". Awesome!")

This will not work, though, as you cannot simply sum strings and floats.

To fix the error, you'll need to explicitly convert the types of your variables. More specifically, you'll need [**str()**](https://docs.python.org/3/library/functions.html#func-str), to convert a value into a string. str(savings), for example, will convert the float savings to a string.

Similar functions such as [**int()**](https://docs.python.org/3/library/functions.html#int), [**float()**](https://docs.python.org/3/library/functions.html#float) and [**bool()**](https://docs.python.org/3/library/functions.html#bool) will help you convert Python values into any type.

##### Instructions

**100 XP**

* Hit Run Code to run the code on the right. Try to understand the error message.
* Fix the code on the right such that the printout runs without errors; use the function [**str()**](https://docs.python.org/3/library/functions.html#func-str) to convert the variables to strings.
* Convert the variable pi\_string to a float and store this float as a new variable, pi\_float.

Script.py  
01 # Definition of savings and result

02 savings = 100

03 result = 100 \* 1.10 \*\* 7

04

05 # Fix the printout

06 print("I started with $" + str(savings) + " and now have $" + str(result) + ". Awesome!")

07

08 # Definition of pi\_string

09 pi\_string = "3.1415926"

10

11 # Convert pi\_string into float: pi\_float

12 pi\_float = float(pi\_string)

IPython Shell  
In [1]: # Definition of savings and result

savings = 100

result = 100 \* 1.10 \*\* 7

# Fix the printout

print("I started with $" + str(savings) + " and now have $" + str(result) + ". Awesome!")

# Definition of pi\_string

pi\_string = "3.1415926"

# Convert pi\_string into float: pi\_float

pi\_float = float(pi\_string)

I started with $100 and now have $194.87171000000012. Awesome!

In [2]:

# Can Python handle everything?

Now that you know something more about combining different sources of information, have a look at the four Python expressions below. Which one of these will throw an error? You can always copy and paste this code in the IPython Shell to find out!

##### Instructions

**50 XP**

##### Possible Answers

* 

"I can add integers, like " + str(5) + " to strings."

* 

"I said " + ("Hey " \* 2) + "Hey!"

* 

"The correct answer to this multiple choice exercise is answer number " + 2

* 

True + False

Submit Answer

IPython Shell  
In [1]: "I can add integers, like " + str(5) + " to strings."

Out[1]: 'I can add integers, like 5 to strings.'

In [2]: "I said " + ("Hey " \* 2) + "Hey!"

Out[2]: 'I said Hey Hey Hey!'

In [3]: "The correct answer to this multiple choice exercise is answer number " + 2

Traceback (most recent call last):

File "<stdin>", line 1, in <module>

"The correct answer to this multiple choice exercise is answer number " + 2

TypeError: Can't convert 'int' object to str implicitly

In [4]: True + False

Out[4]: 1

In [5]: