Intro to Computer Science
CS-UH 1001, Spring 2022

Lecture 2 – Data Types, Variables, Output

Today's Lecture

- Python Data Types
- Operators and Operands
- Variables
- Python Output

Simple Data Types

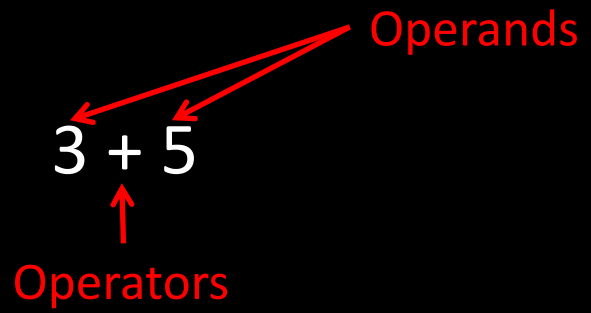
- Python has different data types
- Each data type consists of:
  - A set of values that a data type can hold
  - A set of operations that can be performed on the data type

Type of Data	Python type name	Value examples
Integers	int	
Real numbers	float	
Logical	boolean	
Character strings	str	

- All simple data types are immutable

Operators and Operands

- **Operators** are special symbols that perform an action on one or more operands (data types) Examples: +, -, *, /
- The values the operators is applied to are called **Operands**



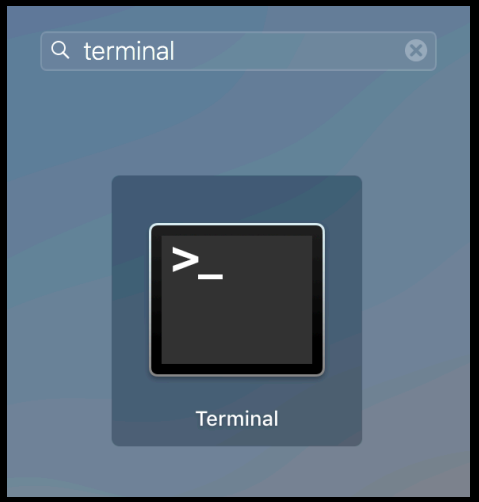
Arithmetic Operators

Operator	Meaning	Examples	Result
+	Addition	2+5	
*	Multiplication	5*2	
-	Subtraction	5-2	
/	Division	6/2 5/2	
**	Power	10**3	
//	Integer division with no decimal value (no remainder)	5//3 14//5	
%	Modulus: remainder of the division	5%3 10%2	

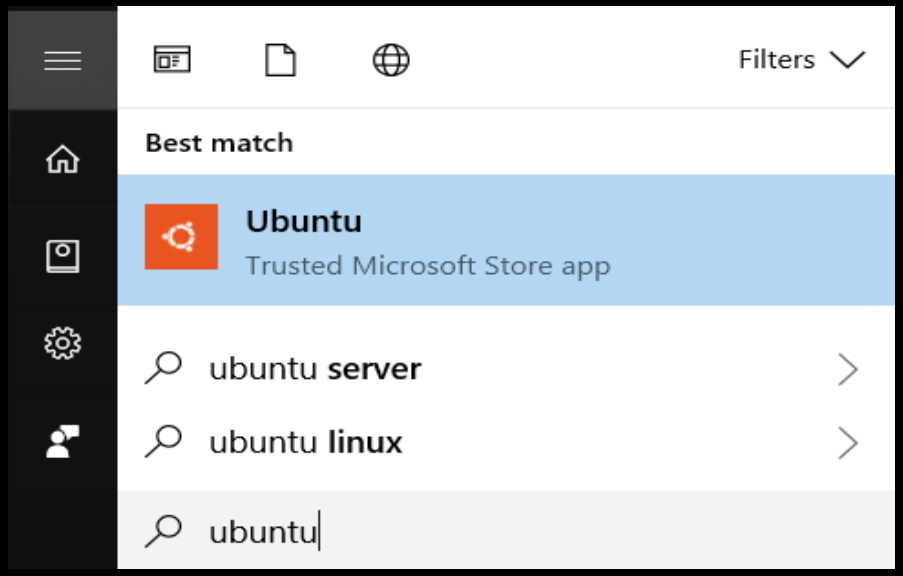
Breakout session I:
Arithmetic Operators

Opening the Command Line

Mac OS

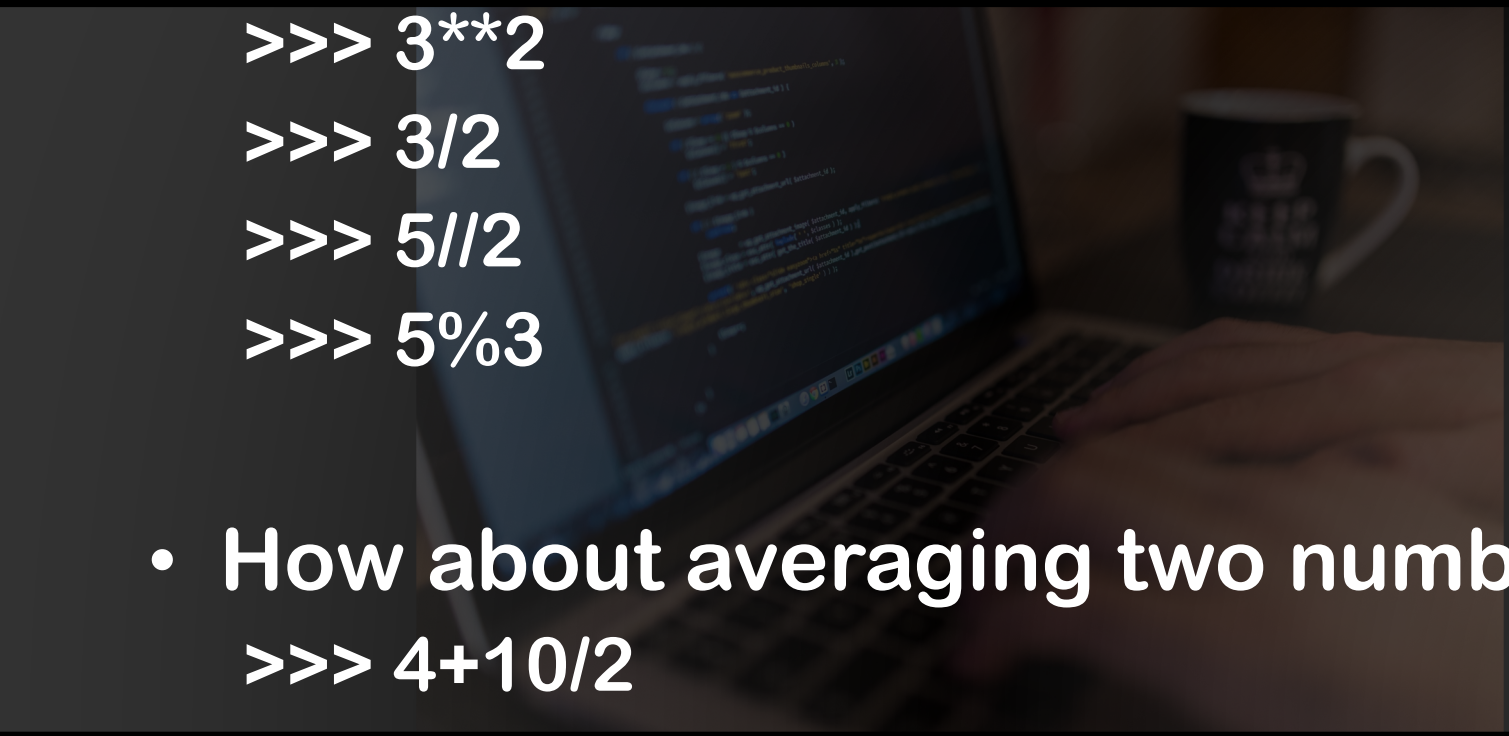


Windows



Enter the interactive mode of Python by typing python3 in your terminal, then execute the following:

```
>>> 3+3
>>> 3**2
>>> 3/2
>>> 5//2
>>> 5%3
```



- How about averaging two numbers?

```
>>> 4+10/2
```

Order of Arithmetic Operations

① Parentheses

② Exponents

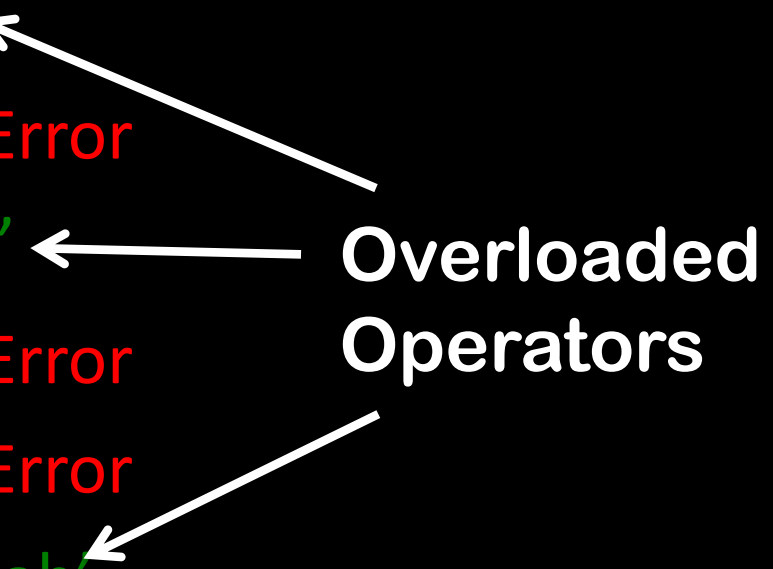
③ Multiplication and Division
Left to Right (5*4-3 is 17 and not 5)

④ Addition and Subtraction
Again Left to Right

Operators for Strings

- **Which one do you think is a valid syntax?**

```
>>> '9'-'1'        TypeError
>>> '1'+'9'        '19'
>>> 1+'9'          TypeError
>>> 'ab'+'cd'      'abcd'
>>> 'ab'-'bc'      TypeError
>>> 'ab'*'a'       TypeError
>>> 'ab'*3         'ababab'
>>> 'ab'/'c'       TypeError
```



Overloaded Operators

- Some operators work differently on different data types:
  - The + (addition) will add two numbers, but concatenates strings
  - The * (multiplication) will multiply two numbers, but performs repetition on strings

- Note:
  You can not subtract (-) or divide (/) strings even if they look like numbers

How to Change Data Types?

- Some data types can be changed by type casting
  - Number as a string can be changed back to a number using **int()** or **float()**

    1 + '9' → Type Error

    1 + int('9') → 10

  - A number can be concatenated with a string by casting the number into string using **str()**

    1 + '9' → Type Error

    str(1) + '9' → '19'

- Note: not all casts are legal

Type Casting

- **What do you think this will give us?**

```
>>> int(4.58)                    4
>>> float(5)                     5.0
>>> str(1) + int('9')            TypeError
>>> str(int('1') + 9)            '10'
>>> int(2 * str(4))              44
>>> int('100.0')                 ValueError
>>> int(float('100.0'))          100
```

Expressions vs. Statements

- Expressions: represents something that gets evaluated and then returns a value

```
>>> 10*(5+4)
90
>>> “Robb” + “ Stark”
Robb Stark
```

- Statements: a unit of code that Python can execute. It will not return a value

```
>>> x = 5
>>>
```

Breakout session I:
Expressions and statements

In the interactive mode of Python, execute the following:

```
>>> x=2+2
>>> x
>>> x*2
>>> x
>>> x=x*2
>>> x
```

Which of them is a statement and which is an expression?

Variables

- Variables are named memory locations that are used to:
  - Bind data to names
  - Store data
  - Manipulate the data and reassign it
- The assignment operator (=) assigns data to variables
- In Python, every data type is an object and contains three pieces of data:
  - Reference (Memory location)
  - Data type
  - Value
- **Python** sets the **variable data type** based on the value that is assigned to it

Variable Assignment

- Variables are created by an assignment operator (=):

	Variable Name	Value	Data Type	Reference
x = 42	x	42	int	94529957049376
price = 10.99	price	10.99	float	48659924322644
name = “Robb”	name	Robb	str	78345092741937
valid = True	valid	1 or True	bool	63223939577737

Variable Names

- Give meaningful variable names!
- If a variable name requires a comment, then the name does not reveal its intent!

- Variable names
  - can only contain letters, numbers or underscores (_)
  - MUST NOT start with a number!
  - MUST NOT be Python keywords

Python Keywords

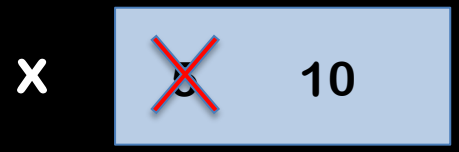
- and
- as
- assert
- break
- class
- continue
- def
- del
- elif
- else
- except
- False
- finally
- for
- from
- global
- if
- import
- in
- is
- lambda
- None
- nonlocal
- not
- or
- pass
- raise
- return
- True
- try
- while
- with
- yield

Naming Conventions

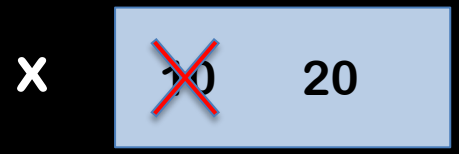
- Common names for temporary variables or counters are:
  - i, j, k, m, and n for integers
  - c, d, and e for characters
- Python recommends:
  - lowercase_separated_by_underscores for variables
  - CAPITALIZED_WITH_UNDERSCORES for constants
  - UpperCamelCase for class names

Variables Re-assignment

- Assignment statements creates variables:
  x = 5
  x = 10



- Assignment statement can have an expression in them:
  x = x +10



Remember the order of the re-assignment:

1. Evaluate the expression
2. Store the value

Obvious, but Common Mistakes

- You must assign something to a variable (create the variable name) before you use (evaluate) it!!!

  Example:
  y = y + 10 ← This will throw an error if y has not been created before

- Don't confuse the assignment operator (single equal sign, =) with the Equality-Test operator (double equal sign, ==)

  Example:
  x = 5
  x == x + 10 ← This will evaluate to False, not 15

Python type()

- Python has a built-in function that tells you the data type of a value or variable

```
>>> type(5)
 <class 'int'>
```

or:

```
>>> x = 5
>>> type(x)
<class 'int'>
```

Python id()

- Python has a built-in function that tells you the memory location of a variable

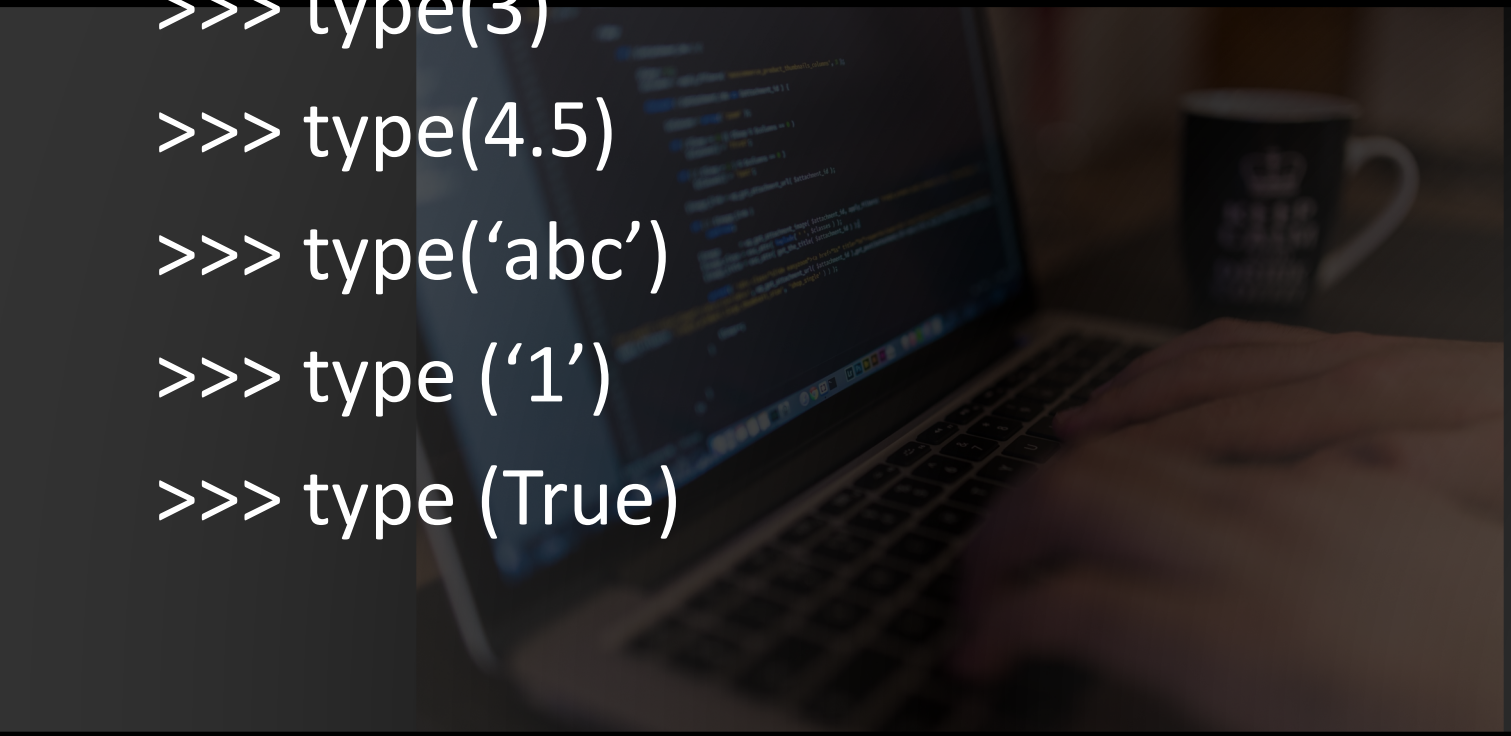
```
>>> x = 5
>>> id(x)
4542294800
```

Breakout session II:
Variables exercises

Data types

Execute the following statements in the interactive mode of Python and observe the output:

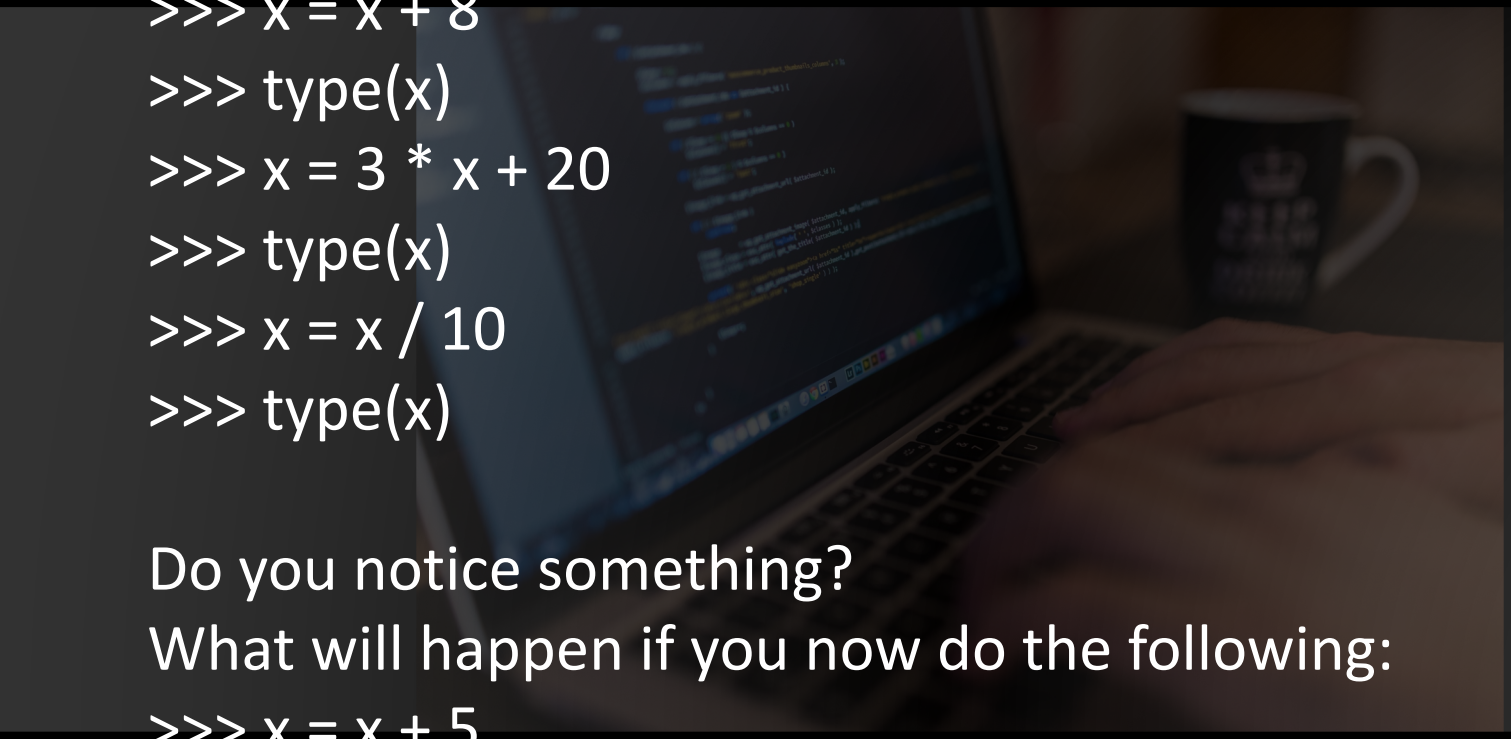
```
>>> type(3)
>>> type(4.5)
>>> type('abc')
>>> type ('1')
>>> type (True)
```



Variable assignments / re-assignments

Execute the following statements:

```
>>> x = 2
>>> type(x)
>>> x = x + 8
>>> type(x)
>>> x = 3 * x + 20
>>> type(x)
>>> x = x / 10
>>> type(x)
```



Do you notice something?

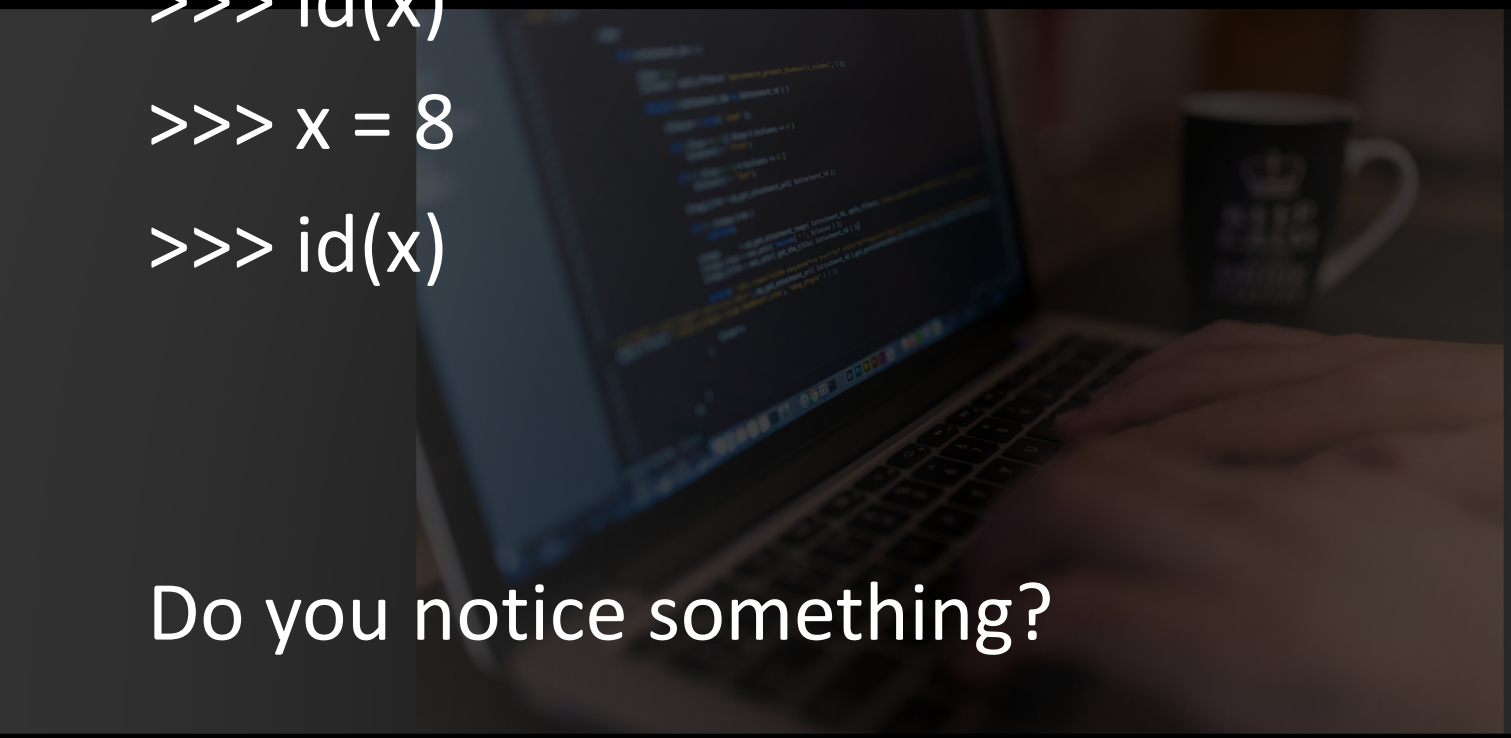
What will happen if you now do the following:

```
>>> x = x + 5
>>> type(x)
```

Variable assignments / re-assignments II

Execute the following statements:

```
>>> x = 2
>>> id(x)
>>> x = 8
>>> id(x)
```



Do you notice something?

Immutability

- All simple data types in Python are immutable (unchangeable):
  - changing its value creates a new memory location holding the same variable
  - immutability is used for internal memory management
  - old/orphaned memory locations will be discarded after some time (garbage collector)

```
# let's take a 15 min break
break_time = True
```

Python Output

- The Python built-in function print() prints a specific message on the screen
  - The message can be a string or any other data type
  - Python will automatically convert the data into a string

Example:

```
>>> print("Hello World!")
Hello World!
```

Breakout session I:
Hello World

The Real World Setup for Programming

- Two parts:
  - The command line (terminal), where you execute your code:
    - python3 filename.py
  - A text editor, where you write your code (the .py file)

Text editor

- You are free to use any text editor you like

- We recommend:
  Sublime Text 3: www.sublimetext.com
  - A very easy and powerful text editor
  - It provides color-coding for Python keywords
  - It's free!

- The next tasks should be coded in a text file rather than through the interactive mode
  - exit the Python interactive mode (>>>) now by typing exit() or CRTL-D
- Create a new text file with Sublime Text, save the text file with a .py extension to your Desktop, for example hello_world.py
  - Remember: avoid Python keywords and spaces in the filename!
- Write the following inside the file and save it
  print("Hello World!")
- Run the code from from the terminal:
  - Navigate to your Desktop (cd Desktop)*
  - Run the program by typing: python3 hello_world.py

*Windows user with Linux subsystem, execute the following command first:
cd /mnt/c/Users/YOURUSERNAME/Desktop

Printing Special Characters

- How about printing the following:
  print("Strings "literally" consist of characters")

- Escape characters are special characters that start with a backslash (\)

Escape Character	Effect
\n	Adds a line break (newline)
\t	Adds a tab
\"	Prints double quote

```
>>> print("Strings \"literally\" consist of characters")
Strings "literally" consist of characters
>>> print("Hello \n World")
Hello
World
```

Python Output

The print() function prints the argument in one line on the screen:

```
>>> print("Hello World!")
Hello World!
```

How about:

```
>>> print("Hello")
>>> print("World!")
Hello
World!
```

By default, every print() function ends with a newline ('\n')

Python Output Line Ending

- You can end a print statement with any character/string using the (end="") parameter

```
>>> print("Hello", end=" ")
>>> print("World!")
Hello World!

>>> print("Hello", end="_")
>>> print("World!")
Hello_World!
```

Long Python Output

- If you want to print several strings at once you can either:
  - use the + operator to concatenate the strings into one long string

```
>>> print("Sunday "+"Monday "+"Tuesday")
Sunday Monday Tuesday
```

  - or use the , separator within the print() function:

```
>>> print("Sunday","Monday","Tuesday")
Sunday Monday Tuesday
```

  - By default, a , will be replaced by a space

Python Output Separator

- If you want to change the space separator into something else, you can define the separator using (sep="") at the end of the print statement

```
>>> print("Sunday","Monday","Tuesday", sep=',')
Sunday,Monday,Tuesday
>>> print("Sunday","Monday","Tuesday", sep='_')
Sunday_Monday_Tuesday
>>> print("Sunday","Monday","Tuesday", sep='\n')
Sunday
Monday
Tuesday
```

Python Variable Output

- You can also print the value of a variable:

```
>>> my_string = “Hello World”
>>> print(my_string)
Hello World

>>> x = 5
>>> print(x)
5
```

Python Variable Output

- You can also print the values of multiple variables at once:

```
>>> month  = “April”
>>> day = 10
>>> print(month,  day)
April 10
```

- How about the following:

```
>>> print(month + day)
>>> print(month + str(day))
April10
```

Type Error

Breakout session I:
Variable output

Simple calculator (ex_2.1.py)

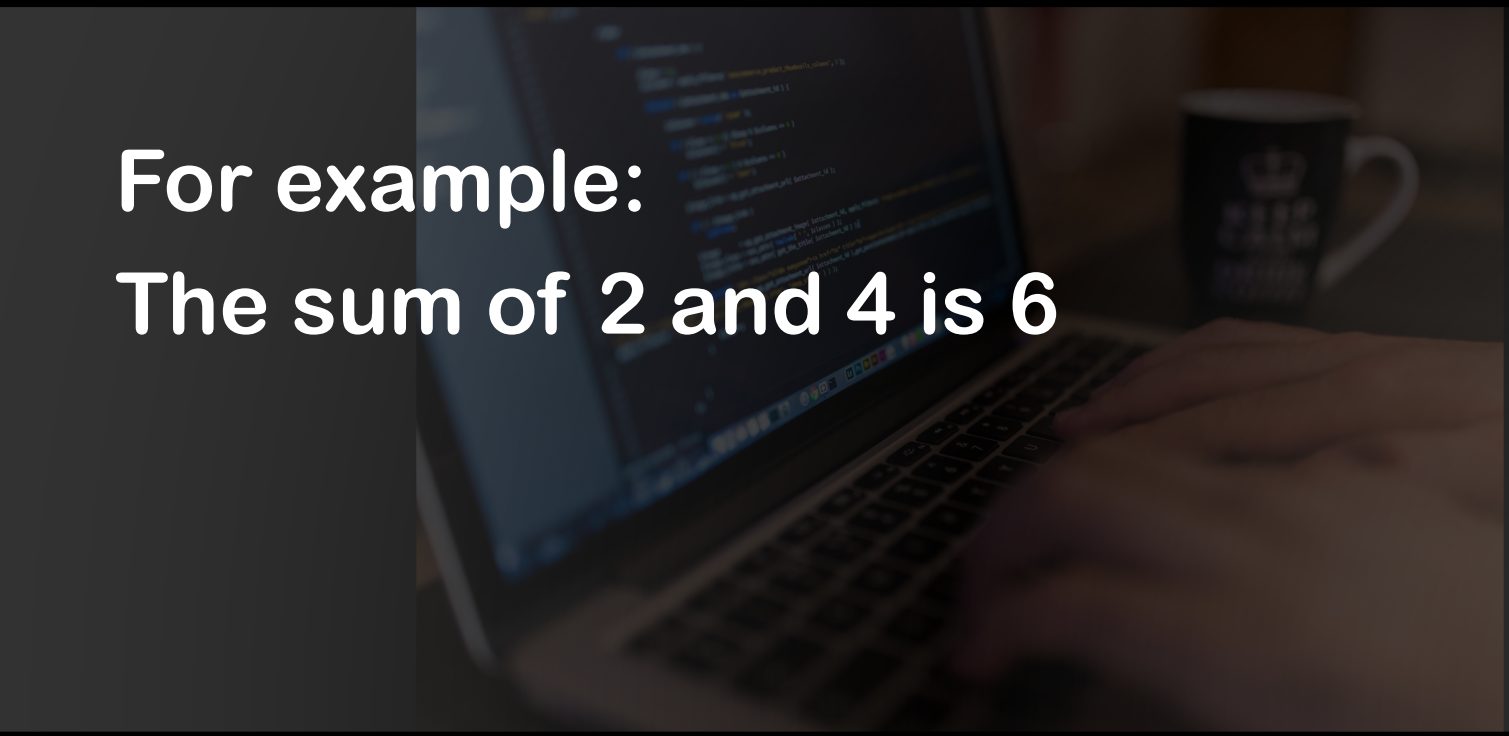
Inside a file, create a simple calculator that sums 2 numbers and prints the result on the screen.

The program should be structured as follows:

- Assign an integer of your choice to a variable, e.g. x
- Assign another integer of your choice to another variable, e.g. y
- Sum both variables and assign the result to another variable , e.g. result
- Print the result

Simple calculator II (ex_2.2.py)

Modify the previous exercise so that the result is printed as follows:



For example:

The sum of 2 and 4 is 6

Next Class

- Characters
- Strings
- String indexing/slicing
- String methods