Lesson 1

Data types, Variables, Conditions

Paola Celio | Pietro Corsi | Sergio Lins

PYTHON BASICS

COMPYLERS

Introduction

How is Python different from other languages?
Why choose Python?

In Python, everything is an object! Python is dynamic and easy to read.
Vast amount of libraries available online!

Data types

Declaring a variable:

In Python, declaring variables is simple and straightforward. Since the variable will point to a value (as an object) and not to a memory address, there is no need to allocate a specific block of memory to it beforehand, by actually binding it to a data type and declaring. Keep in mind that everything is CASE SENSITIVE

In the line A = 42, the operator “=” should be read as “is set to” or “read as”.

Why? Because A can change its value and TYPE during the execution of the program.

Data types

Declaring a variable:

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

Data types

What happens when you tell your computer to memorize something?

Data types

Numeric	Text	Logical
Integer (int)	**string (str)**	**boolean (bool)**
float point (float)		**True or False**
complex		

Numeric data types

Operation between different numeric data types:

int `+` float = float

float `\` int = float

int `*` complex = complex

float `-` complex = complex

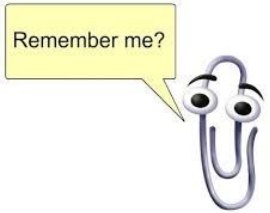
When performing operations, the most complex data type will prevail.

Numeric data types

```
In [5]: A = 2.0
        B = 3
        C = 3+2j

        print(type(A),type(B),type(C))
```

```
<class 'float'> <class 'int'> <class 'complex'>
```



print() is a function. We will get to it later. What is does is just printing things on your screen. We need it to control our outputs.

```
print("Results:")
print(A+B,A+C,B+C)
print(B+A,C+A,+C+B)

print(type(A+B),type(A+C),type(B+C))
print(type(B+A),type(C+A),type(C+B))
```

```
Results:
5.0 (5+2j) (6+2j)
5.0 (5+2j) (6+2j)
<class 'float'> <class 'complex'> <class 'complex'>
<class 'float'> <class 'complex'> <class 'complex'>
```

Numeric data types

So, what would be the type of variable C below?

A = 5

B = 3

C = A/B

Numeric data types

Fractions are represented as decimals.

Python dynamically allocates memory, therefore the block “reserved” for the variable is not fixed as in other languages.

C will be defined by pointing to the quotient of A and B, (which is equal to 1.66667) and so it will be declared as float type

```
A = 5
B = 3
print(type(A))
print(type(B))
C = A/B
print(type(C))
C
```

```
<class 'int'>
<class 'int'>
<class 'float'>

1.6666666666666667
```

Numeric data types

And how do we FORCE python to maintain the variable as specific type?

int( ), float( ) and complex( ) are some tools we can use to change the data type.

a = int(3.14)
a will be read as 3

b = float(1)
b will be read as 1.0

If we take our previous example and change it a bit:

A = 5

B = 3

C = int(A/B)

C will be read as 1 instead of 1.666667

Numeric data types (operators)

+	sum	+=	x += 5	x = x + 5
-	subtract	-=	x -= 5	x = x - 5
/	divide	/=	x /= 5	x = x / 5
//	floor division	//=	x //= 5	x = x // 5
*	multiply	*=	x *= 5	x = x * 5
**	pow	**=	x **= 5	x = x ** 5
%	modulus	%=	x %= 5	x = x % 5

Numeric data types

Operators priority:

The priority will follow the mathematical rules, *i.e.,* exponential operations will be performed before any product or division, which in turn are performed before any sum or subtraction.

(**), (*, /, //, %), (+,-)

Exercise

1. Set your variables;
2. Write your equation;
3. Run.

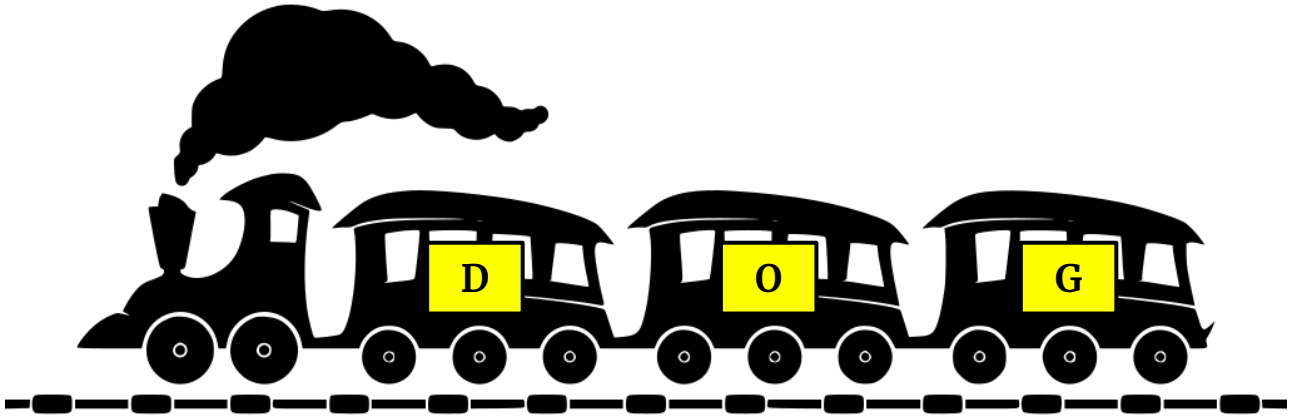
```
d = a*b+a**b-5*a+c/a//b
```

```
d = a*b+a**b-5*a+c/a//b
```

$$d = (a.b) + \left(a^b\right) - (5a) + \left(\left\lfloor \frac{c/a}{b} \right\rfloor\right)$$

Text data type

What is the difference between a character and a string?



Text data type

Strings are sequences, so they have length and intrinsic methods.

Again, in Python, everything is an object!

- Defining a string
- Escape sequence (\) and special characters (new line, tab and return)
- Think of a *typewriter*

```
a = "I am a string"
b = "Me too, but with special characters \\ \' and \""

print(b)
```

```
Me too, but with special characters \ ' and "
```

Text data type

```
string1 = "I am making a new line at the end.\n"
string2 = "\tI can start with tabulation!"
string3 = "I can do the \r (almost) the same as string1."

print(string1)
print(string2)
print(string3)
```

```
I am making a new line at the end.

        I can start with tabulation!
 (almost) the same as string1.
```

Text data type

Strings can also be numeric.

a = “42” is different from a = 42

But, a = “42” is the same as a = str(42)

If you try to perform the following operation, what will the output be?

a, b = “42”,“01”

a + b = ?

Text data type

Strings can be concatenated. This operation can be done using the operand “+”

```
a, b, c = "bread","olives","wine"

print(a + b + c)
print(a + " " + b)
```

```
breadoliveswine
bread olives
```

Text data type

Concatenating can be very useful. How to write a PATH, for example?

String1 = “C:\\”

String2 = “my folder\\”

String3 = “totally_not_a_virus.exe”

file_path = String1 + String2 + String3

file_path is then read as: “C:\\my folder\\totally_not_a_virus.exe”

Useful text methods

variable.replace(“what should be replaced”, “what will replace it”)

variable.split(“when to split”)

variable.join(another_string)

Why using methods instead of writing the code yourself?

```
string1 = "Forza Roma!"
string1 = string1.replace("Roma","Lazio")
print(string1)

string2 = "The sentence above is a lie."
string2 = string2.split(" ")
print(string2)
```

```
Forza Lazio!
['The', 'sentence', 'above', 'is', 'a', 'lie.']
```

Boolean (logical) data type

This data type will contain truth values, which can be only True or False.

They come in handy when comparing values or establishing conditions. For example, when you want to know if a number is greater than other, equal to something, if a string starts with a specific character, or just if something exists.

Comparison operators

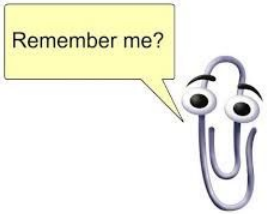
```
a, b = 2, 3.45
a > b
```

```
False
```

```
a, b = 2, 3.45
c = a > b
print(c)
```

```
False
```

==	is equal to
!=	is different to
<	lesser than
>	greater than
>=	greater or equal to
<=	lesser or equal to



You can attribute the truth values to a variable. This allows you to check it later on your program.

```
text1 = "bread"
text2 = "foods"

text1 != text2
```

```
True
```

Comparison operators

Chaining comparison operators.

As mathematical operators, one can chain different comparison operators, like in the expression: $0 < variable \leq 6$

The operations follows always from left to right in the case of comparison, so the program will check first if 0 is lesser than *variable* and then if *variable* is equal to or lesser than 6. If both statements are True, the output will be True. If only one statement if True, the output will be False.

Logical operators

in and not in are operators that apply to iterables only. Remember that strings are sequences of characters. You cannot verify if 5 is in 45, for example, since they are integers

is	is equal to
is not	is different to
in	contains it
not in	does not contain
not	opposite to
and	Logical AND
or	Logical OR

Logical operators

```
no1 = 42.05
no2 = 33

no1 < no2 or no1-no2 < 20
```

```
True
```

```
text1 = "bread"
text2 = "foods"

"w" in text1 and "z" not in text2
```

```
False
```

```
text1 = "bread"
text2 = "foods"

not "w" in text1 and "z" not in text2
```

```
True
```

```
text1 = "bread"
text2 = "foods"

text1 is text2 or "b" not in text2
```

```
True
```

Logical operators

Logical operators can also be chained, as comparison operators. They are read from left to right.

```
fruit_a = "apple"
fruit_b = "ananas"
fruit_c = "peach"
fruit_salad = fruit_a.join(fruit_b)

fruit_a is not fruit_c not in fruit_salad
```

```
True
```

Logical operators

Note that both “types” of operators can be combined together. Take the statement below for example:

(“ma” in “Roma” == True)

False

The statement above is read as (“ma” in “Roma”) and (“Roma” == True), which is obviously False.

True

Each statement returns a boolean value. They can be chained only because what is compared are True and False values.

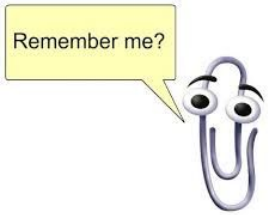
Logical operators

If no logical operator “or” is included in the chain, each statement is chained with “and” in between them as it was seen before. When “or” is added, if the first statement is already True, the chain ends there.

```
a = True
b = False
c = "banana"
c or a is c
```

```
'banana'
```

In this example, c is True. Therefore the statement *a is c* is not read and “banana” is returned as output



A numerical value 0 or an empty string are considered False values.

```
a = True
b = False
c = ""
c or a is c
```

```
False
```

Exercise

Return True or False based if the rest of the division of 42 by 39 contains the floor division of 3.14 to the power of 2 by 4.

Solution

```
>>>print(str(int(3.14**2//3)) in str(42%39))

>>>True
```

(more) Useful text methods

```
variable.isalpha()          #check if all characters in the string are alphabetic
variable.isdigit()          #test if string is a digit
variable.isupper()          #test if string is uppercase only
variable.islower()          #test if string is lowercase only
variable.isspace()          #test if string is a space
variable.endswith("s")      #test if string endswith an s
variable.startswith("H")    #test if string startswith H
```

```
text1 = "42 almonds"
text1.startswith("4")
```

```
True
```

```
text1 = "42 almonds"
text1.isdigit()
```

```
False
```

Conditions

We know how to compare things, but now we want to do something according to the situation (or conditions)

*"**If** it is sunny **or** cloudy, I'll leave my umbrella at home **and** wear shorts.*
***Otherwise**, I'll take my umbrella with me **and** wear trousers."*

Conditions

True and False values (booleans) will allow you to verify certain conditions and execute commands according to them.

```
sunny = False
cloudy = True

if sunny or cloudy:
    print("Wear shorts and leave umbrella")
else:
    print("I will take an umbrella and wear trousers")
```

```
Wear shorts and leave umbrella
```

Conditions

Note that if it is raining or not has no effect whatsoever in the conditional. What is being verified is if its is sunny or cloudy. The fact that it is not sunny and not cloudy doesn’t mean it is raining.

Conditions

Checking more than one condition:

```
sunny, cloudy, raining, snowing = False, False, True, False

if sunny or cloudy:
    print("Wear trousers and leave umbrella")
elif raining:
    print("I will take an umbrella and wear trousers")
elif snowing:
    print("I'm taking my snowboard out!")
else:
    print("If it is not snowing, raining, cloudy or sunny, there is something really weird happening outside")
```

```
I will take an umbrella and wear trousers
```

Pay attention to indentation!

Conditions

```
a = 1.0
b = 2
c = 3.14
d = 4.20
e = 5
f = 42

if f % d > -a-c**a or f-c/a+b >= e*b:
    print(c)
else:
    g = a+b+c+d+e
    print(g)
```

```
a, b, c, d, e = 3, 6, 7, 3, 2
average = (a+b+d+e)/4

if c > average and average >= 0:
    c = average
else:
    c = c

print(c)
```

```
3.5
```

Conditions

If we are not sure if the program can verify the condition, for example, if a variable has not been set so far or if the type of operation can result in an error, Python has a very useful structure.

Try/Except

Conditions

Let’s say you want to tell your program to divide a value by the result of a chain of complex operations that can result in the denominator being assigned the value 0.

We all know that a division by zero is a mathematical error, and if you try to divide anything by zero, your program will fail. In this case, you need to check the value of the variable BEFORE telling the program to perform the operation.

Conditions

```
if denominator != 0:
    c = numerator/denominator
else:
    pass
```

```
try:
    c = numerator/denominator
except:
    raise ZeroDivisionError
```


https://docs.python.org/3/tutorial/errors.html

Conditions

Try and except are very useful when you need to perform operations you are unsure that can be done or when checking all the conditions for the operation to be possible is too complex or not feasible.

Create/remove folders or files, execute functions or methods.

User input

The input() Python function.

We already saw print() several times so far, which serves only to output text to the console screen. There is an opposite function, called input(), which allows the user to input data with the keyboard. To keep it simple, we are not getting into too much detail.

```
variable = input()
or
variable = input("Some text")
```

User input

input() will get whatever was typed and point it to the associated variable.

The variable will be of the type STRING

Exercise

1. Get 2 numbers and a mathematical operator (+,-,*,/) from user input.
2. First number must be greater than 0 and lesser than 10
3. Second number must be greater than 10 and lesser than 20
4. Return the input operation between the first and second numbers.

Homework (optional)

1. Install OpenCV library;
2. Using what was taught in this lesson, get the filename by input;
3. Try to open the file with “image = cv2.imread(“FILE PATH”)”
4. Print the variable associated to the image you just opened.

https://github.com/linssab/python_basics_roma3