Intro to programming 1

Henri Vandendriessche
henri.vandendriessche@ens.fr

2022-09-20

Context

- inscription on the moodle: https://moodle.u-paris.fr/enrol/index.php?id=43747#section-0

Context

- inscription on the moodle: https://moodle.u-paris.fr/enrol/index.php?id=43747#section-0
- Why programming ?

Context

- inscription on the moodle: https://moodle.u-paris.fr/enrol/index.php?id=43747#section-0
- Why programming ?
  - Important for your independence as young researcher

Context

- inscription on the moodle: https://moodle.u-paris.fr/enrol/index.php?id=43747#section-0
- Why programming ?
  - Important for your independence as young researcher
  - develops the iterative thinking and computational thinking

Context

- inscription on the moodle: https://moodle.u-paris.fr/enrol/index.php?id=43747#section-0
- Why programming ?
  - Important for your independence as young researcher
  - develops the iterative thinking and computational thinking
  - Good for problem solving in general

Context

- inscription on the moodle: https://moodle.u-paris.fr/enrol/index.php?id=43747#section-0
- Why programming ?
  - Important for your independence as young researcher
  - develops the iterative thinking and computational thinking
  - Good for problem solving in general
  - In general knowing how to code is an important skill

Context

- inscription on the moodle: https://moodle.u-paris.fr/enrol/index.php?id=43747#section-0
- Why programming ?
  - Important for your independence as young researcher
  - develops the iterative thinking and computational thinking
  - Good for problem solving in general
  - In general knowing how to code is an important skill
- Why python ?

Context

- inscription on the moodle: https://moodle.u-paris.fr/enrol/index.php?id=43747#section-0
- Why programming ?
  - Important for your independence as young researcher
  - develops the iterative thinking and computational thinking
  - Good for problem solving in general
  - In general knowing how to code is an important skill
- Why python ?
  - It's free and open

Context

- inscription on the moodle: https://moodle.u-paris.fr/enrol/index.php?id=43747#section-0
- Why programming ?
  - Important for your independence as young researcher
  - develops the iterative thinking and computational thinking
  - Good for problem solving in general
  - In general knowing how to code is an important skill
- Why python ?
  - It's free and open
  - Versatility

Context

- inscription on the moodle: https://moodle.u-paris.fr/enrol/index.php?id=43747#section-0
- Why programming ?
  - Important for your independence as young researcher
  - develops the iterative thinking and computational thinking
  - Good for problem solving in general
  - In general knowing how to code is an important skill
- Why python ?
  - It's free and open
  - Versatility
  - Popularity

Context

- inscription on the moodle: https://moodle.u-paris.fr/enrol/index.php?id=43747#section-0
- Why programming ?
  - Important for your independence as young researcher
  - develops the iterative thinking and computational thinking
  - Good for problem solving in general
  - In general knowing how to code is an important skill
- Why python ?
  - It's free and open
  - Versatility
  - Popularity
  - Simplicity and robustness

Context

- inscription on the moodle: https://moodle.u-paris.fr/enrol/index.php?id=43747#section-0
- Why programming ?
  - Important for your independence as young researcher
  - develops the iterative thinking and computational thinking
  - Good for problem solving in general
  - In general knowing how to code is an important skill
- Why python ?
  - It's free and open
  - Versatility
  - Popularity
  - Simplicity and robustness
- Why programming in python for cognitive sciences ?

Context

- inscription on the moodle: https://moodle.u-paris.fr/enrol/index.php?id=43747#section-0
- Why programming ?
  - Important for your independence as young researcher
  - develops the iterative thinking and computational thinking
  - Good for problem solving in general
  - In general knowing how to code is an important skill
- Why python ?
  - It's free and open
  - Versatility
  - Popularity
  - Simplicity and robustness
- Why programming in python for cognitive sciences ?
  - Good for data science, modelling and experimental design at the same time

Context

- inscription on the moodle: https://moodle.u-paris.fr/enrol/index.php?id=43747#section-0
- Why programming ?
  - Important for your independence as young researcher
  - develops the iterative thinking and computational thinking
  - Good for problem solving in general
  - In general knowing how to code is an important skill
- Why python ?
  - It's free and open
  - Versatility
  - Popularity
  - Simplicity and robustness
- Why programming in python for cognitive sciences ?
  - Good for data science, modelling and experimental design at the same time
  - Great scientific community using it and developing tools (https://www.frontiersin.org/articles/10.3389/fninf.2015.00011/full)

Survival rules for programming

- 1 Try by yourself before looking for solutions

Survival rules for programming

- 1 Try by yourself before looking for solutions
- 2 Internet is your best friend

Survival rules for programming

- 1 Try by yourself before looking for solutions
- 2 Internet is your best friend
- 3 Read the manual

Survival rules for programming

- 1 Try by yourself before looking for solutions
- 2 Internet is your best friend
- 3 Read the manual
- 4 There is always a manual

Survival rules for programming

- 1 Try by yourself before looking for solutions
- 2 Internet is your best friend
- 3 Read the manual
- 4 There is always a manual
- 5 Have you read the fucking manual ?

Survival rules for programming

- 1 Try by yourself before looking for solutions
- 2 Internet is your best friend
- 3 Read the manual
- 4 There is always a manual
- 5 Have you read the fucking manual ?
- 6 Not yet ? Then read it

Survival rules for programming

- 1 Try by yourself before looking for solutions
- 2 Internet is your best friend
- 3 Read the manual
- 4 There is always a manual
- 5 Have you read the fucking manual ?
- 6 Not yet ? Then read it
- 7 Always read the error message

Resources 1/2

Books & ebooks

- Gérard Swinnen Apprendre à Programmer avec Python 3 (5e edition)
  http://inforef.be/swi/python.htm
- Al Sweigart How to automate the boring stuff with Python (2e edition)
  https://automatetheboringstuff.com/
- Al Sweigart Invent Your Own Computer Games with Python (4e edition)
  http://inventwithpython.com/invent4thed/

Resources 2/2

Online course & Mooc

Openclassrooms

- https://openclassrooms.com/fr/courses/7168871-apprenez-les-bases-du-langage-python
- https://openclassrooms.com/en/courses/6902811-learn-python-basics

Mooc de l'Inria

- https://www.my-mooc.com/fr/mooc/python-des-fondamentaux-a-lutilisation-du-langage/

Websites

- https://pythontutor.com/ (Visualize and step by step code execution)

Terminal cheatsheet

- Bash commands to navigate directories

Terminal cheatsheet

- Bash commands to navigate directories
- Useful commands:

Terminal cheatsheet

- Bash commands to navigate directories
- Useful commands:
  - **pwd** Print Working Directory. Print the path of of the current directory

Terminal cheatsheet

- Bash commands to navigate directories
- Useful commands:
  - **pwd** Print Working Directory. Print the path of of the current directory
  - **ls /folder** list all files of the current directory

Terminal cheatsheet

- Bash commands to navigate directories
- Useful commands:
  - **pwd** Print Working Directory. Print the path of of the current directory
  - **ls /folder** list all files of the current directory
  - **cd /folder1/folder2** moving into folder1 and folder2 at once.

Terminal cheatsheet

- Bash commands to navigate directories
- Useful commands:
  - **pwd** Print Working Directory. Print the path of of the current directory
  - **ls /folder** list all files of the current directory
  - **cd /folder1/folder2** moving into folder1 and folder2 at once.
  - **cd ..** moving out of a directory

Terminal cheatsheet

- Bash commands to navigate directories
- Useful commands:
  - **pwd** Print Working Directory. Print the path of of the current directory
  - **ls /folder** list all files of the current directory
  - **cd /folder1/folder2** moving into folder1 and folder2 at once.
  - **cd ..** moving out of a directory
- "Tab" to use the auto-completion

Terminal cheatsheet

- Bash commands to navigate directories
- Useful commands:
  - **pwd** Print Working Directory. Print the path of of the current directory
  - **ls /folder** list all files of the current directory
  - **cd /folder1/folder2** moving into folder1 and folder2 at once.
  - **cd ..** moving out of a directory
- "Tab" to use the auto-completion
- Many more bash commands to use. . .

Writing and running a program with python

- Open sublime
- Write:

```
print("Hello !")
```

```
## Hello !
```

- Save the file as *hello.py*
- Open a terminal and navigate to the folder where you have saved your program and run the command: `python hello.py`

Variables and data in python I

- Declaring variables

```
x = 10
print(x)
```

```
## 10
```

Variables and data in python II

- Naming variables

Variables and data in python II

- Naming variables
  - Use alphanumeric characters and underscores (but no number nor underscore as first character)

Variables and data in python II

- Naming variables
  - Use alphanumeric characters and underscores (but no number nor underscore as first character)
  - Can't use a reserved keywords

Variables and data in python II

- Naming variables
  - Use alphanumeric characters and underscores (but no number nor underscore as first character)
  - Can't use a reserved keywords
  - Can contain digits but not only digits and an not start with a digit

Variables and data in python II

- Naming variables
  - Use alphanumeric characters and underscores (but no number nor underscore as first character)
  - Can't use a reserved keywords
  - Can contain digits but not only digits and an not start with a digit
  - Variable are case-sensitive: upper-case and lower-case make a difference

Variables and data in python II

- Naming variables
  - Use alphanumeric characters and underscores (but no number nor underscore as first character)
  - Can't use a reserved keywords
  - Can contain digits but not only digits and an not start with a digit
  - Variable are case-sensitive: upper-case and lower-case make a difference
  - There is no limit of length

```
x = 10
print(x)
```

```
## 10
```

```
abc = 2
print(abc)
```

```
## 2
```

```
_ = 5
print(_)
```

```
## 5
```

This work

Variables and data in python III

- Naming variables

```
1x = 10
print(x)
```

```
## invalid syntax (<string>, line 1)
```

```
a bc = 2
print(abc)
```

```
## invalid syntax (<string>, line 1)
```

```
t-t = 10
print(t-t)
```

```
## cannot assign to operator (<string>, line 1)
```

This doesn't work

Variables and data in python III

- Naming variables
  - Use alphanumeric characters and underscores (but no number nor underscore as first character)

```
1x = 10
print(x)
```

```
## invalid syntax (<string>, line 1)
```

```
a bc = 2
print(abc)
```

```
## invalid syntax (<string>, line 1)
```

```
t-t = 10
print(t-t)
```

```
## cannot assign to operator (<string>, line 1)
```

This doesn't work

Variables and data in python III

- Naming variables
  - Use alphanumeric characters and underscores (but no number nor underscore as first character)
  - Can't use a reserved keywords

```
1x = 10
print(x)
```

```
## invalid syntax (<string>, line 1)
```

```
a bc = 2
print(abc)
```

```
## invalid syntax (<string>, line 1)
```

```
t-t = 10
print(t-t)
```

```
## cannot assign to operator (<string>, line 1)
```

This doesn't work

Variables and data in python III

- Naming variables
  - Use alphanumeric characters and underscores (but no number nor underscore as first character)
  - Can't use a reserved keywords
  - Can contain digits but not only digits and an not start with a digit

```
1x = 10
print(x)
```

```
## invalid syntax (<string>, line 1)
```

```
a bc = 2
print(abc)
```

```
## invalid syntax (<string>, line 1)
```

```
t-t = 10
print(t-t)
```

```
## cannot assign to operator (<string>, line 1)
```

This doesn't work

Variables and data in python III

- Naming variables
  - Use alphanumeric characters and underscores (but no number nor underscore as first character)
  - Can't use a reserved keywords
  - Can contain digits but not only digits and an not start with a digit
  - Variable are case-sensitive: upper-case and lower-case make a difference

```
1x = 10
print(x)
```

```
## invalid syntax (<string>, line 1)
```

```
a bc = 2
print(abc)
```

```
## invalid syntax (<string>, line 1)
```

```
t-t = 10
print(t-t)
```

```
## cannot assign to operator (<string>, line 1)
```

This doesn't work

Variables and data in python III

- Naming variables
  - Use alphanumeric characters and underscores (but no number nor underscore as first character)
  - Can't use a reserved keywords
  - Can contain digits but not only digits and an not start with a digit
  - Variable are case-sensitive: upper-case and lower-case make a difference
  - There is no limit of length

```
1x = 10
print(x)
```

```
## invalid syntax (<string>, line 1)
```

```
a bc = 2
print(abc)
```

```
## invalid syntax (<string>, line 1)
```

```
t-t = 10
print(t-t)
```

```
## cannot assign to operator (<string>, line 1)
```

This doesn't work

Variables and data in python IV

- Different types of variables: Primitive types

Variables and data in python IV

- Different types of variables: Primitive types
  - integers

```
x = 10
print(type(x))
```

```
## <class 'int'>
```

Variables and data in python IV

- Different types of variables: Primitive types
  - integers

```
x = 10
print(type(x))
```

```
## <class 'int'>
```

  - float

```
y = 5.5
print(type(y))
```

```
## <class 'float'>
```

Variables and data in python IV

- Different types of variables: Primitive types
  - integers

```
x = 10
print(type(x))
```

```
## <class 'int'>
```

  - float

```
y = 5.5
print(type(y))
```

```
## <class 'float'>
```

  - string

```
z = "test"
print(type(z))
```

```
## <class 'str'>
```

Variables and data in python IV

- Different types of variables: Primitive types
  - integers

```
x = 10
print(type(x))
```

```
## <class 'int'>
```

  - float

```
y = 5.5
print(type(y))
```

```
## <class 'float'>
```

  - string

```
z = "test"
print(type(z))
```

```
## <class 'str'>
```

  - boolean

```
w = True
print(type(w))
```

```
## <class 'bool'>
```

Numeric data et operations 1/2

- Differences between integers and floats
- Arithmetic operators + - * / %
- Exercise : calculate and print the result of this operation

$$\frac{15}{3+2} - (\frac{\frac{100}{4}}{5}) * 2$$

Numeric data et operations 2/2

- Differences between integers and floats
- Arithmetic operators + - * / %
- Exercise : calculate and print the result of this operation

$$\frac{15}{3+2} - (\frac{\frac{100}{4}}{5}) * 2$$

```
x = 15/(3+2) - (100/4/5)*2
print(x)
```

```
## -7.0
```

Strings 1/3

- Used to store text (most of the time)
- Strings are declared with " " or ' '. Be careful when you want to print a string already stored in a variable. Examples:

```
animal = "Dog"
course = 'Intro to programming in python'

print(animal)
```

```
## Dog
```

```
print("animal")
```

```
## animal
```

Strings 2/3

- String can easily be concatenated the operator $+$

```
greetings = "Hello"
presentation = "My name is"
name = "Henri"
print( greetings + presentation + name)
```

```
## HelloMy name isHenri
```

- Exercise: insert white space between the words.

Strings 2/3

- String can easily be concatenated the operator $+$

```
greetings = "Hello"
presentation = "My name is"
name = "Henri"
print( greetings + presentation + name)
```

```
## HelloMy name isHenri
```

- Exercise: insert white space between the words.

```
greetings = "Hello"
presentation = "My name is"
name = "Henri"
print( greetings + ' ' + presentation + ' ' + name)
```

```
## Hello My name is Henri
```

```
space = ' '
print(greetings + space + presentation + space + name)
```

```
## Hello My name is Henri
```

Strings 3/3

- Functions working with strings:

Strings 3/3

- Functions working with strings:
  - **len()** calculate the length of a function. Example:

```
greetings = "Hello"
print(len(greetings))
```

```
## 5
```

Strings 3/3

- Functions working with strings:
  - **len()** calculate the length of a function. Example:

```
greetings = "Hello"
print(len(greetings))
```

```
## 5
```

  - **replace()** can replace a pattern in a string. Example:

```
greetings = "Hello"
print(greetings)
```

```
## Hello
```

```
print(greetings.replace('e','a'))
```

```
## Hallo
```

Strings 3/3

- Functions working with strings:
  - **len()** calculate the length of a function. Example:

```
greetings = "Hello"
print(len(greetings))
```

```
## 5
```

  - **replace()** can replace a pattern in a string. Example:

```
greetings = "Hello"
print(greetings)
```

```
## Hello
```

```
print(greetings.replace('e','a'))
```

```
## Hallo
```

  - ...

Booleans

- A boolean is a pretty simple variable but sometimes complex to deal with

Booleans

- A boolean is a pretty simple variable but sometimes complex to deal with
- It can have only two values **True** or **False**

Booleans

- A boolean is a pretty simple variable but sometimes complex to deal with
- It can have only two values **True** or **False**
- It is used usually to store the truth value of logic. Example:

```
Is_it_Weekend = False
print(Is_it_Weekend)
```

```
## False
```

Program flow 1/4

- A program normally executes sequentially from top to bottom

Program flow 1/4

- A program normally executes sequentially from top to bottom
- But some instructions can change the flow. For examples *if* and *for* loops

Program flow 1/4

- A program normally executes sequentially from top to bottom
- But some instructions can change the flow. For examples *if* and *for* loops
- **if** will test for a conditional comparison. If a condition is true then we can execute some lines of codes if it's false we will not execute those lines of codes (but why not others). Example 1:

```
Weekend = False
if Weekend == True :
  print("Let's do nothing and chill")
else:
  print("Time to go to work")
```

```
## Time to go to work
```

Program flow 1/4

- A program normally executes sequentially from top to bottom

Program flow 1/4

- A program normally executes sequentially from top to bottom
- But some instructions can change the flow. For examples *if* and *for* loops

Program flow 1/4

- A program normally executes sequentially from top to bottom
- But some instructions can change the flow. For examples *if* and *for* loops
- **if** will test for a conditional comparison. If a condition is true then we can execute some lines of codes if it's false we will not execute those lines of codes (but why not others). Example2 :

```
string = "This string is long but not that long"

if len(string) < 10:
  print("This string has less than 10 character")
elif len(string) < 20:
  print("This string has less than 20 character")
elif len(string) >30:
  print("That string is too long for me...")
```

```
## That string is too long for me...
```

Program flow 2/4

- You can also define other alternatives with **elif**

Program flow 2/4

- You can also define other alternatives with **elif**
- you can make use different ways to make your comparison: - **and** - **or** - **not**

Program flow 2/4

- You can also define other alternatives with **elif**
- you can make use different ways to make your comparison: - **and** - **or** - **not**
- You can as well use comparative expressions: - < - > - <= - == tests for equality - **!=** tests for inequality

Program flow 2/4

- You can also define other alternatives with **elif**
- you can make use different ways to make your comparison: - **and** - **or** - **not**
- You can as well use comparative expressions: - < - > - <= - == tests for equality - != tests for inequality
- Be careful = is not the same as ==

```
age = 30 #(affectation)

age == 30 #(equality comparison that returns TRUE if correct)
```

```
## True
```

```
age == 31
```

```
## False
```

Program flow 3/4

- **for** loops iterate one or a set of operations several times.

Example 1:

```
for x in range(10):
  print(x)
```

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

Program flow 3/4

- **for** loops iterate one or a set of operations several times.

Example 1:

```
for x in range(10):
  print(x)
```

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

Example 2:

```
for i in range(5,10):
  print(i)
```

```
## 5
## 6
```

Program flow 4/4

- With **for** loops you see that your program flow is not always unilateral from the top to the bottom of your script

Program flow 4/4

- With **for** loops you see that your program flow is not always unilateral from the top to the bottom of your script
- Let's play a small game. Download https://github.com/chrplr/PCBS/blob/master/games/human-guess-a-number.py

Program flow 4/4

- With **for** loops you see that your program flow is not always unilateral from the top to the bottom of your script
- Let's play a small game. Download https://github.com/chrplr/PCBS/blob/master/games/human-guess-a-number.py
- Then go on http://pythontutor.com/ and paste the code of the game.

Program flow 4/4

- With **for** loops you see that your program flow is not always unilateral from the top to the bottom of your script
- Let's play a small game. Download https://github.com/chrplr/PCBS/blob/master/games/human-guess-a-number.py
- Then go on http://pythontutor.com/ and paste the code of the game.
- Look at the program flow to see how it jumps from one code section to another

Exercises

- Exercise 1: Write code that prints the string “All work and no play makes Jack a dull boy” 50 times
- Exercise 2: Write code that prints the square of all integers between 1 and 100 using range
- Exercise 3: Write code that browses the integer from 0 to 100 but only prints the number 1, 50 and 100
- Exercise 4: Write code that prints only even numbers between 0 and 100
- Exercise 5: Write code that computes the factorial of an integer (no function, no recursion, just a loop)