Intro to programming 1

Henri Vandendriessche
henri.vandendriessche@ens.fr

2023-09-19

Context

- inscription on the moodle: https://moodle.u-paris.fr/course/view.php?id=43747

Context

- inscription on the moodle: https://moodle.u-paris.fr/course/view.php?id=43747
- and discord: https://discord.gg/9hKDQmRu

Context

- inscription on the moodle: https://moodle.u-paris.fr/course/view.php?id=43747
- and discord: https://discord.gg/9hKDQmRu
- Why programming ?

Context

- inscription on the moodle: https://moodle.u-paris.fr/course/view.php?id=43747
- and discord: https://discord.gg/9hKDQmRu
- Why programming ?
  - **Independence**: Empowers young researchers for independent work.

Context

- inscription on the moodle: https://moodle.u-paris.fr/course/view.php?id=43747
- and discord: https://discord.gg/9hKDQmRu
- Why programming ?
  - **Independence**: Empowers young researchers for independent work.
  - **Thinking Skills**: Develops iterative, computational, and problem-solving thinking.

Context

- inscription on the moodle: https://moodle.u-paris.fr/course/view.php?id=43747
- and discord: https://discord.gg/9hKDQmRu
- Why programming ?
  - **Independence**: Empowers young researchers for independent work.
  - **Thinking Skills**: Develops iterative, computational, and problem-solving thinking.
  - **Essential Skill**: Coding is a vital skill in the digital age.

Context

- inscription on the moodle: https://moodle.u-paris.fr/course/view.php?id=43747
- and discord: https://discord.gg/9hKDQmRu
- Why programming ?
  - **Independence**: Empowers young researchers for independent work.
  - **Thinking Skills**: Develops iterative, computational, and problem-solving thinking.
  - **Essential Skill**: Coding is a vital skill in the digital age.
- Why python ?

Context

- inscription on the moodle: https://moodle.u-paris.fr/course/view.php?id=43747
- and discord: https://discord.gg/9hKDQmRu
- Why programming ?
  - **Independence**: Empowers young researchers for independent work.
  - **Thinking Skills**: Develops iterative, computational, and problem-solving thinking.
  - **Essential Skill**: Coding is a vital skill in the digital age.
- Why python ?
  - **Accessibility**: Free and open-source, eliminating cost barriers.

Context

- inscription on the moodle: https://moodle.u-paris.fr/course/view.php?id=43747
- and discord: https://discord.gg/9hKDQmRu
- Why programming ?
  - **Independence**: Empowers young researchers for independent work.
  - **Thinking Skills**: Develops iterative, computational, and problem-solving thinking.
  - **Essential Skill**: Coding is a vital skill in the digital age.
- Why python ?
  - **Accessibility**: Free and open-source, eliminating cost barriers.
  - **Versatility**: A unified language for data analysis, modeling, and more.

Context

- inscription on the moodle: https://moodle.u-paris.fr/course/view.php?id=43747
- and discord: https://discord.gg/9hKDQmRu
- Why programming ?
  - **Independence**: Empowers young researchers for independent work.
  - **Thinking Skills**: Develops iterative, computational, and problem-solving thinking.
  - **Essential Skill**: Coding is a vital skill in the digital age.
- Why python ?
  - **Accessibility**: Free and open-source, eliminating cost barriers.
  - **Versatility**: A unified language for data analysis, modeling, and more.
  - **Community**: Supported by a large and active user community.

Context

- inscription on the moodle: https://moodle.u-paris.fr/course/view.php?id=43747
- and discord: https://discord.gg/9hKDQmRu
- Why programming ?
  - **Independence**: Empowers young researchers for independent work.
  - **Thinking Skills**: Develops iterative, computational, and problem-solving thinking.
  - **Essential Skill**: Coding is a vital skill in the digital age.
- Why python ?
  - **Accessibility**: Free and open-source, eliminating cost barriers.
  - **Versatility**: A unified language for data analysis, modeling, and more.
  - **Community**: Supported by a large and active user community.
- Why programming in python for cognitive sciences ?

Context

- inscription on the moodle: https://moodle.u-paris.fr/course/view.php?id=43747
- and discord: https://discord.gg/9hKDQmRu
- Why programming ?
  - **Independence**: Empowers young researchers for independent work.
  - **Thinking Skills**: Develops iterative, computational, and problem-solving thinking.
  - **Essential Skill**: Coding is a vital skill in the digital age.
- Why python ?
  - **Accessibility**: Free and open-source, eliminating cost barriers.
  - **Versatility**: A unified language for data analysis, modeling, and more.
  - **Community**: Supported by a large and active user community.
- Why programming in python for cognitive sciences ?
  - **Comprehensive Toolkit**: Ideal for data science, modeling, and experimental design.

Context

- inscription on the moodle: https://moodle.u-paris.fr/course/view.php?id=43747
- and discord: https://discord.gg/9hKDQmRu
- Why programming ?
  - **Independence**: Empowers young researchers for independent work.
  - **Thinking Skills**: Develops iterative, computational, and problem-solving thinking.
  - **Essential Skill**: Coding is a vital skill in the digital age.
- Why python ?
  - **Accessibility**: Free and open-source, eliminating cost barriers.
  - **Versatility**: A unified language for data analysis, modeling, and more.
  - **Community**: Supported by a large and active user community.
- Why programming in python for cognitive sciences ?
  - **Comprehensive Toolkit**: Ideal for data science, modeling, and experimental design.
  - **Research Community**: Thriving community with specialized tools for cognitive sciences. (https://www.frontiersin.org/articles/10.3389/fninf.2015.00011/full)

Survival rules for programming

- 1 Try by yourself before seeking solutions.

Survival rules for programming

- 1 Try by yourself before seeking solutions.
- 2 Internet is your best friend.

Survival rules for programming

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

Survival rules for programming

- 1 Try by yourself before seeking 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 seeking 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 seeking 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 seeking 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 review error messages carefully.

Survival rules for programming

- 1 Try by yourself before seeking 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 review error messages carefully.
- 8 Only then, consider asking ChatGPT for help.

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/
- Allen B. Downey: Think Python http://greenteapress.com/thinkpython2/

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/

Udemy

- Udemy: Python programming for absolute beginners: https://www.udemy.com/python-programming-for-absolute-beginners/

Code Academy

- Code Academy: Learn Python module: https://www.codecademy.com/learn/learn- python

Other

- Python 3: des fondamentaux aux concepts avancés du langage: https://lms.fun-mooc.fr/

Terminal cheat sheet

- Bash commands to navigate directories

```
pwd
```

```
ls folder
```

```
cd folder1/subfolder2
```

```
cd ..
```

```
cd ~
```

Terminal cheat sheet

- Bash commands to navigate directories
  - Print Working Directory. Print the path of the current directory

```
pwd
```

```
ls folder
```

```
cd folder1/subfolder2
```

```
cd ..
```

```
cd ~
```

Terminal cheat sheet

- Bash commands to navigate directories
  - Print Working Directory. Print the path of the current directory

```
pwd
```

  - List all files of the current directory

```
ls folder
```

```
cd folder1/subfolder2
```

```
cd ..
```

```
cd ~
```

Terminal cheat sheet

- Bash commands to navigate directories
  - Print Working Directory. Print the path of the current directory

```
pwd
```

  - List all files of the current directory

```
ls folder
```

  - Moving into folder1 and subfolder2 at once.

```
cd folder1/subfolder2
```

```
cd ..
```

```
cd ~
```

Terminal cheat sheet

- Bash commands to navigate directories
  - Print Working Directory. Print the path of the current directory

```
pwd
```

  - List all files of the current directory

```
ls folder
```

  - Moving into folder1 and subfolder2 at once.

```
cd folder1/subfolder2
```

  - Moving out of a directory

```
cd ..
```

```
cd ~
```

Terminal cheat sheet

- Bash commands to navigate directories
  - Print Working Directory. Print the path of the current directory

```
pwd
```

  - List all files of the current directory

```
ls folder
```

  - Moving into folder1 and subfolder2 at once.

```
cd folder1/subfolder2
```

  - Moving out of a directory

```
cd ..
```

  - Going back to the root directory

```
cd ~
```

Terminal cheat sheet

- Bash commands to navigate directories
  - Print Working Directory. Print the path of the current directory

```
pwd
```

  - List all files of the current directory

```
ls folder
```

  - Moving into folder1 and subfolder2 at once.

```
cd folder1/subfolder2
```

  - Moving out of a directory

```
cd ..
```

  - Going back to the root directory

```
cd ~
```

- “Tab” to use the auto-completion

Terminal cheat sheet

- Bash commands to navigate directories
  - Print Working Directory. Print the path of the current directory

```
pwd
```

  - List all files of the current directory

```
ls folder
```

  - Moving into folder1 and subfolder2 at once.

```
cd folder1/subfolder2
```

  - Moving out of a directory

```
cd ..
```

  - Going back to the root directory

```
cd ~
```

- "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

- A variable is a named container for a particular set of bits or type of data (like integer, float, string etc. . .)

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

```
## 10
```

Variables and data in python I

- A variable is a named container for a particular set of bits or type of data (like integer, float, string etc. . .)
- 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 numbers or underscores as the first character).

Variables and data in python II

- Naming variables
  - Use alphanumeric characters and underscores (but no numbers or underscores as the first character).
  - Avoid using reserved keywords.

Variables and data in python II

- Naming variables
  - Use alphanumeric characters and underscores (but no numbers or underscores as the first character).
  - Avoid using reserved keywords.
  - Variables can contain digits but must not consist solely of digits and cannot start with a digit.

Variables and data in python II

- Naming variables
  - Use alphanumeric characters and underscores (but no numbers or underscores as the first character).
  - Avoid using reserved keywords.
  - Variables can contain digits but must not consist solely of digits and cannot start with a digit.
  - Variables are case-sensitive; uppercase and lowercase letters are distinct.

Variables and data in python II

- Naming variables
  - Use alphanumeric characters and underscores (but no numbers or underscores as the first character).
  - Avoid using reserved keywords.
  - Variables can contain digits but must not consist solely of digits and cannot start with a digit.
  - Variables are case-sensitive; uppercase and lowercase letters are distinct.
  - There is no length limit for variables.

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

```
## 10
```

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

```
## 2
```

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

```
## 5
```

Variables and data in python III

- Naming variables
  - Use alphanumeric characters and underscores (but no numbers or underscores as the first character).
  - Avoid using reserved keywords.
  - Variables can contain digits but must not consist solely of digits and cannot start with a digit.
  - Variables are case-sensitive; uppercase and lowercase letters are distinct.
  - There is no length limit for variables.

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

```
## invalid decimal literal (<string>, line 1)
```

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

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

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

- Use to store text (most of the time)
- Strings are declared with double quotation marks (" ") or single quotation marks (' '). Be careful when printing 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 with 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 with 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 relatively simple variable but can sometimes be challenging to work with.

Booleans

- A boolean is a relatively simple variable but can sometimes be challenging to work with.
- It can have only two values **True** or **False**

Booleans

- A boolean is a relatively simple variable but can sometimes be challenging to work 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 typically executes sequentially, moving from top to bottom.

Program flow 1/4

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

Program flow 1/4

- A program typically executes sequentially, moving from top to bottom.
- But some instructions can change the flow. For examples *if* and *for* loops
- **if** will test a conditional comparison. If the condition is true, we can execute certain lines of code; if it's false, those lines of code may not be executed (though other code may still be executed). 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 typically executes sequentially, moving from top to bottom.

Program flow 1/4

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

Program flow 1/4

- A program typically executes sequentially, moving from top to bottom.
- But some instructions can change the flow. For examples *if* and *for* loops
- **if** will test a conditional comparison. If the condition is true, we can execute certain lines of code; if it's false, those lines of code may not be executed (though other code may still be executed). 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 multiple times.

Example 1:

```
for x in range(5):
  print(x)
```

```
## 0
## 1
## 2
## 3
## 4
```

Program flow 3/4

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

Example 1:

```
for x in range(5):
  print(x)
```

```
## 0
## 1
## 2
## 3
## 4
```

Example 2:

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

```
## 5
## 6
## 7
## 8
## 9
```

Program flow 4/4

- With **for** loops, you can observe that your program flow is not always unidirectional from the top to the bottom of your script.

Program flow 4/4

- With **for** loops, you can observe that your program flow is not always unidirectional 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 can observe that your program flow is not always unidirectional 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 can observe that your program flow is not always unidirectional 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.
- Examine the program flow to observe how it transitions from one code section to another.

Clarification on for loops

- A **for** loop is used to iterate over the elements of a sequence (such as a string, tuple, set, list, or dictionary) or another iterable object.:

```
list1 = [1,2,3,0]
for x in list1:
  print(x)
```

```
## 1
## 2
## 3
## 0
```

```
list2 = [1,2,3,0]
for x in range(0,len(list2)):
  print(list2[x])
```

```
## 1
## 2
## 3
## 0
```

Clarification on for loops

- A **for** loop is used to iterate over the elements of a sequence (such as a string, tuple, set, list, or dictionary) or another iterable object.:
- It differs from other **for** keyword in other programming languages and works more like an iterator.

```
list1 = [1,2,3,0]
for x in list1:
  print(x)
```

```
## 1
## 2
## 3
## 0
```

```
list2 = [1,2,3,0]
for x in range(0,len(list2)):
  print(list2[x])
```

```
## 1
## 2
## 3
## 0
```

Clarification on for loops

- A **for** loop is used to iterate over the elements of a sequence (such as a string, tuple, set, list, or dictionary) or another iterable object.:
- It differs from other **for** keyword in other programming languages and works more like an iterator.

```
list1 = [1,2,3,0]
for x in list1:
  print(x)
```

```
## 1
## 2
## 3
## 0
```

- It is almost similar to (which is much closer to the "traditional" for loop in programming)

```
list2 = [1,2,3,0]
for x in range(0,len(list2)):
  print(list2[x])
```

```
## 1
## 2
## 3
## 0
```

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 squares of all integers between 1 and 100 using the *range* function.
- Exercise 3: Write code that iterates through integers from 0 to 100 but only prints the numbers 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 using a loop (without using a function or recursion).