intro-to-programming-2

Henri Vandendriessche

28/09/2021

Reminder

- 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. . .

So far

- Python
- Variables
- Data types:
  - integer
  - float
  - string
  - boolean
- If and For loops:
  - syntax
  - indentation

Reading advice

To complete what we're going to see today.

- https://automatetheboringstuff.com/2e/chapter4/
- https://automatetheboringstuff.com/2e/chapter5/

Today

- While loop
- Other python data types for collections of data type
  - list
  - set
  - tuple
  - dictionary
- Random numbers and number choices
- Excercices

While loop 1/3

- **While** loop will keep looping all over again an ending instruction is met.
- As the **for** loop incorporate a specified number of execution, the **while** loop is best suited for unknown or very large number of loop iterations.
- As for **if** and **for**, **while** has a syntax and need indentation for the following lines to be included in the process
- The two key features of a while loop are:
  - the output condition
  - the increment statement

While loop 2/3

- Example:

```
i = 1
while i < 4: #  output condition
  print(i)
  i += 1      # increment statement
```

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

```
## Which is technically the same as
for i in range(1,4):
  print(i)
```

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

While loop 3/3

- **While** loop will test if the output condition is True and if not it will execute the code and execute the increment statement
- If one of those two conditions are not correctly specified you'll encounter an error or an infinite loop...

```
i = 1
while i < 6: #  output condition
  print(i)

while i != 6:
  print(i)
  x += 2
```

Lists 1/3

- A list is a collection of related objects
- It's declared with brackets [] with a comma between two objects

```
Years_France_won_worldcup = [1998, 2018]
print(Years_France_won_worldcup)
```

```
## [1998, 2018]
```

```
dog_breeds = ["golden", "corgi", "Bulldog", "Husky", "Beagle"]
dog_breeds2 = ["golden" "corgi" "Bulldog" "Husky" "Beagle"]
print(dog_breeds)
```

```
## ['golden', 'corgi', 'Bulldog', 'Husky', 'Beagle']
```

```
print(dog_breeds2)
```

```
## ['goldencorgiBulldogHuskyBeagle']
```

```
random_data_type_collection = [ 1, True, "Cats", 3.14]
print(random_data_type_collection)
```

```
## [1, True, 'Cats', 3.14]
```

Lists 2/3

- Access element in a list through its index which is the same to access characters in a string as in a list

```
prog_language = ["python", "R", "C", "java", "Go", "Rust"]
print(prog_language[0])
```

```
## python
```

```
print(prog_language[-1])
```

```
## Rust
```

```
programming_language = "python"
print(programming_language[0])
```

```
## p
```

Lists 3/3

- some functions
  - append()
  - remove()
  - pop()
  - sort()
  - len()

```
prog_language = ["python", "R", "C", "java", "Go", "Rust"]
prog_language.append("html")
prog_language.append("PHP")
print(prog_language)
```

```
## ['python', 'R', 'C', 'java', 'Go', 'Rust', 'html', 'PHP']
```

```
prog_language.remove("html")

len(prog_language)
```

```
## 7
```

```
prog_language.sort()
print(prog_language)
```

```
## ['C', 'Go', 'PHP', 'R', 'Rust', 'java', 'python']
```

Tuples 1/3

- Very similar to Lists and is used for data collection
- Declared with () instead of square bracket

```
date_covid_shots = ("21-04-15", "21-05-18", "21-09-20")

print(type(date_covid_shots)) # type function is very important
```

```
## <class 'tuple'>
```

```
print(date_covid_shots[1])  # Accessible as list with index with []
```

```
## 21-05-18
```

```
print(len(date_covid_shots))
```

```
## 3
```

Tuples 2/3

- In contrast to lists, they are immutable and can't be modified.

```
date_covid_shots = ("21-04-15", "21-05-18", "21-09-20")
date_covid_shots.append("21-09-27")
```

- You can't change the order of items neither modify the value of an item
- Tuples are best suited when you need ordered lists that would never change
  - If you want to code a calendar : days and years can be coded as tuples as they would not change but are ordered.

Tuples 3/3

- Note that you could combine lists and tuples

```
Cocktails = [("Cosmo","5€"),("Daiquiri","7€"),("B52","6€")]
Cocktails.append(("Mojito","7€"))

print(Cocktails)
```

```
## [('Cosmo', '5€'), ('Daiquiri', '7€'), ('B52', '6€'), ('Mojito', '7€')]
```

- NB: you can also declare a tuple that way

```
date_covid_shots = tuple(["21-04-15", "21-05-18", "21-09-20"])
# in this line you transform a list into a tuple
print(type(date_covid_shots))
```

```
## <class 'tuple'>
```

Sets

- Very close to lists but are unordered unindexed and do not allow duplicate value BUT are mutable
- Declared with {}

```
fruit_I_like = {"apple", "pineapple", "peach"}
print(type(fruit_I_like))
```

```
## <class 'set'>
```

```
print(fruit_I_like)
```

```
## {'peach', 'pineapple', 'apple'}
```

```
fruit_I_like.add("strawberry")
"strawberry" in fruit_I_like # Check if a fruit is in my set
```

```
## True
```

```
fruit_I_like.remove("apple")
print(fruit_I_like)
```

```
## {'strawberry', 'peach', 'pineapple'}
```

- But can't add/remove items in a set or access to item with index. Following instructions should throw an error
  `fruit_I_like[0]`
  `fruit_I_like.append("banana")`

Dictionaries 1/

- Data structure that uses data in key-value pairs.
- Each items of a dictionary is a key-value pair
- Each key has to be unique
- Declared in a very specific way

  `my_dictionary = { "key1" : value1, "key2": value2 ...}`

```
PCBS = {
  "Name" : "PCBS",
  "Teacher" : "Christophe Pallier",
  "Teacher assistant1" : "Cedric",
  "Teacher assistant2" : "Henri",
  "Day": "Tuesday",
  "Duration" : 3,
  "Mandatory" : False}

print(PCBS)
```

```
## {'Name': 'PCBS', 'Teacher': 'Christophe Pallier', 'Teacher assistant1': 'Ced
```

Dictionaries 2/3 Create a Dictionary

- You have several methods to create a dictionary:
- Examples

```
PCBS = {}
PCBS["Name"] = "PCBS"
PCBS["Teacher"] = "Christophe Pallier"
PCBS["Teacher assistant1"] = "Cedric"
PCBS["Teacher assistant2"] = "Henri"
PCBS["Day"] = "Tuesday"
PCBS["Duration"] = 3
PCBS["Mandatory"] = False
print(PCBS)
```

```
## {'Name': 'PCBS', 'Teacher': 'Christophe Pallier', 'Teacher assistant1': 'Ced
```

- Which is exactly the same as

```
PCBS = dict()
PCBS["Name"] = "PCBS"
PCBS["Teacher"] = "Christophe Pallier"
PCBS["Teacher assistant1"] = "Cedric"
PCBS["Teacher assistant2"] = "Henri"
PCBS["Day"]= "Tuesday"
PCBS["Duration"] = 3
PCBS["Mandatory"] = False
print(PCBS)
```

Dictionaries 3/3 Use common operation

- Access to a a key-value pair
- Add a key-value pair
- Delete a key-value pair
- Check for specific key existence

```
PCBS = {  "Name" : "PCBS", "Teacher" : "Christophe Pallier",
"Teacher assistant1" : "Cedric", "Teacher assistant2" : "Henri",
"Day": "Tuesday", "Duration" : 3, "Mandatory" : False}
PCBS['Day']
```

```
## 'Tuesday'
```

```
PCBS["starting time"] = "13h30"
PCBS.pop("Teacher assistant2")
```

```
## 'Henri'
```

```
print(PCBS)
```

```
## {'Name': 'PCBS', 'Teacher': 'Christophe Pallier', 'Teacher assistant1': 'Ced
```

Summary on Python collections (~ Arrays)

	List	tuple	Set	Dictionary
Mutable	✔	✘	✔	✔
Ordered	✔	✔	✘	✔
Indexing	✔	✔	✘	✔
Duplicate elements	✔	✔	✘	⚠ values can be duplicated Keys can't
Can be created using	list()	tuple()	set()	dict()

Python module - example of **Random** 1/4

- Python incorporates in its standard library a multitude of modules for a variety of subjects and problem (network, text processing, mathematics, file and directory access, cryptography. . .)

https://docs.python.org/3/library/index.html

- The standard library include in particular a specific module for random (pseudo-random) number generation

https://docs.python.org/3/library/random.html

Python module - example of **Random** 2/4

- Several ways to import a python module

```
import random # import random
int_list =[1,2,3]
random.shuffle(int_list)# from that object you have to access all the functions
print(int_list)
```

```
## [2, 3, 1]
```

```
import random as rand # import random using a custom local name
rand.shuffle(int_list) # from that object you have to access all the functions
print(int_list)
```

```
## [1, 2, 3]
```

```
from random import shuffle,randint,choice # import only needed function
shuffle(int_list) # use the function directly without object before
print(int_list)
```

```
## [2, 1, 3]
```

```
from random import * # import all the functions bundled inside Random at once
shuffle(int_list)
print(int_list)
```

```
## [3, 1, 2]
```

Python module - example of **Random** 3/4

```
from random import *

print(randint(1, 100))    # Pick a random integer between 1 and 100.
```

```
## 44
```

```
print(uniform(1, 100))    # Pick a random float between 1 and 100.

# prints a random value from the list
```

```
## 43.811205668393015
```

```
list1 = [1, 2, 3, 4, 5, 6]
print(choice(list1))
```

```
## 1
```

```
items = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
y = sample(items, 4)    # Pick 4 random items from the list
print(y)
```

```
## [5, 9, 7, 10]
```

Python module - example of **Random** 4/4

```
# using randrange() to generate in range from 20
# to 50. The last parameter 3 is step size to skip
# three numbers when selecting.
print("A random number from range is : ", end="")
```

```
## A random number from range is :
```

```
print(randrange(20, 50, 3))
```

```
## 35
```

Excercices 1/2

- Exercice 1: Lists: - Given a list of numbers, print their sum - Given a list of numbers, print their product - Given a list of numbers, print the sum of their squares - Given a list of numbers, print the largest one. - Given a list of numbers, print the second largest
- Exercice 2: Given a list of words, count the number of times each word appears in the list (using dictionary)
- Exercice 3: Lottery pick. Generate 100 random lottery tickets (one ticket is a sequence of 5 digits) and pick one winner out of it.
- Exercice 4: write a program that generates a random 10 character long password including 6 letters with 2 of them uppercase, 1 digit and 1 special symbol.
- Exercice 5: Monte Carlo estimation of Pi: one way to estimate the value of the pi is to generate a large number of random points in the unit square and see how many fall within the unit circle; their proportion is an estimate of the area of the circle. See https://academo.org/demos/estimating-pi-monte-carlo. Implement the proposed algorithm to estimate the value of pi.
- Exercice 6: Write a program that prints the first N rows of Pascal's triangle (see https://www.youtube.com/watch?v=XMriWTvPXHI).