PYTHON MODULE IMPORT EXIT AND A SHORT PROGRAM

CSC 109 - Introduction to programming in Python - Summer 2023

Table of Contents

1. Importing modules

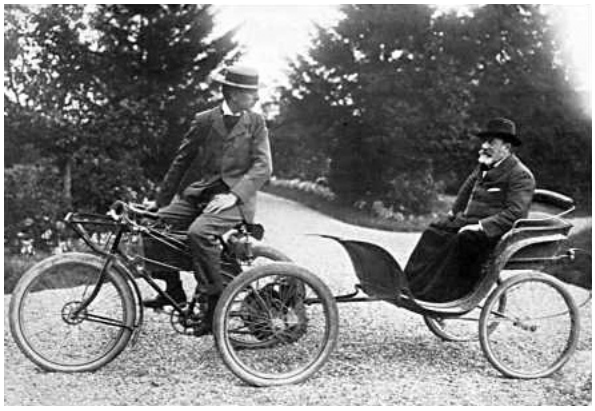

Figure 1: De Dion-Bouton tridcycle towing a passenger (ca. 1919)

In this section, you'll learn how to `import` useful functions, find out details about your computer's `os`, get `random` numbers and the current `time`, and look at the system.

- You've seen several built-in functions: `print`, `len`, `str` etc.

```
print("hello")
print(len("hello"))
print(len(str(100000000000)))
```

```
You typed __PYTHON_EL_eval("try:\n    with open('c:/Users/BIRKEN~1/AppData/Local/Temp/babel-KmZkAf/python-KUfiRk') as __org
```

- Additional functions are contained in the *standard library*, which is a large collection of *modules*, each of which contain *methods* or functions (link).
- Later in this course, we'll look at exceptions, strings, math modules, system services, regular expressions from the standard library.

2. Importing with `import`

- Before you can use the methods in a module, you must `import` it with:
  1. the keyword `import` followed by
  2. the name of the module, e.g. `os` for *operating system*
  3. Optionally more module names separated by comma
- To call a module's method, you must prefix the method name with the module name like so: `os.getenv`, `datetime.time` etc.

- **Exercise:** to see *environment* variables and their values, `import os` and get individual variables with `os.getenv` - do this locally in an IDLE instance:

```
import os
print(os.getenv('OS'))    # the operating system
print(os.getenv('HOME'))    # your home directory
print(os.getenv('USERNAME'))  # your user name
print(os.getenv('NUMBER_OF_PROCESSORS'))  # CPUs of your computer
```

```
You typed __PYTHON_EL_eval("try:\n    with open('c:/Users/BIRKEN~1/AppData/Local/Temp/babel-KmZkAf/python-zhQIPp') as __org
```

- You should see something like this (with different values):

```
>>> import os
>>> os.getenv('HOME')
'C:\\Users\\birkenkrahe'
>>> os.getenv('USERNAME')
'Birkenkrahe'
>>> os.getenv('NUMBER_OF_PROCESSORS')
'16'
```

Figure 2: Environment variables with os.getenv() in IDLE

3. Seeing what's been imported with `sys`

- To see ALL environment variables, you can use a method inside a method of `os` and loop over two variables simultaneously:

```
for key, value in os.environ.items():
    print(key,value)
```

```
You typed __PYTHON_EL_eval("try:\n    with open('c:/Users/BIRKEN~1/AppData/Local/Temp/babel-KmZkAf/python-2ISNXR') as __org
```

- **Exercise:** To see all modules that are imported, you can use the method `sys.modules`:
  1. `import` the `sys` module
  2. return the length of `sys.modules` with `len`
  3. do this in IDLE for your local PC
  4. compare with Google Colab in an online notebook

- Solution: `sys.modules` is a *dictionary*, which is a Python data structure. You can find this out with `type`.

```
import sys
print(len(sys.modules))
print(type(sys.modules))
```

```
You typed __PYTHON_EL_eval("try:\n    with open('c:/Users/BIRKEN~1/AppData/Local/Temp/babel-KmZkAf/python-BJol8K') as __org
```

4. Random numbers with `random`

- **Exercise:** You'll need random numbers for a few of our projects:
  1. import the `random` module
  2. create a `for` loop of `range` 10
  3. consult the `help` on `random.randint` in Colab.
  4. in the clause, print 10 random numbers in the range [1,10] with the `randint` method from the `random` module.

- Solution: the `help` function delivers concise parameter information.

```
import random
print(help(random.randint))
for i in range(10):
    print(random.randint(1,10))
```

```
You typed __PYTHON_EL_eval("try:\n    with open('c:/Users/BIRKEN~1/AppData/Local/Temp/babel-KmZkAf/python-u8CaXs') as __org
```

- You can step through this code at autbor.com/printrandom/: notice how the `import` command leads to the creation of a *module instance* as an *object*. This is a preview of Python's OOP properties.

5. Breaking Python with `random`

- When you `import` modules, you need to be careful not to overwrite their names with names of your own Python scripts: don't use script names like `random.py`, `os.py` or `sys.py`. Same for *method* names.

- Let's create an empty file `random.py` and see what happens when we load `random` in the presence of that file (you can do it in a REPL):

  1. To create an empty file, enter `touch random.py`
  2. Open the Python shell with `python`
  3. Enter `import random`

4. Look at `help(random.randint)`
5. Exit with `exit()`
6. Remove `random.py` with `del random.py`

```
$ ls random*
  random.py
$ python
Python 3.11.1 (tags/v3.11.1:a7a450f, Dec  6 2022, 19:58:39)
Type "help", "copyright", "credits" or "license" for more information.
>>> import random
>>> help(random.randint)
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
AttributeError: module 'random' has no attribute 'randint'
>>>
```

Figure 3: Breaking Python with an empty file random.py

6. Importing specific methods with `from`

- An alternative importing method for modules is `from`, which allows you to use methods without module prefix - do this on the Python shell:

```
from random import randint
print(randint(1,10))
print(sample([1,2,3,4],2))
```

```
You typed __PYTHON_EL_eval("try:\n    with open('c:/Users/BIRKEN~1/AppData/Local/Temp/babel-KmZkAf/python-aZYZ6U') as __org
```

- In the last call to `random.sample` (drawing 2 out of a group of 4 numbers with replacement), `sample` was not known because it was not loaded:

```
from random import sample
print(sample([1,2,3,4],2))
```

```
You typed __PYTHON_EL_eval("try:\n    with open('c:/Users/BIRKEN~1/AppData/Local/Temp/babel-KmZkAf/python-7jQ5oI') as __org
```

7. Ending programs early with `sys.exit`

- Programs terminate when the program execution reaches the last instruction.
- You can force termination before the last instruction by calling `sys.exit()` inside your program.

- Save the following code as a program `exitExample.py` in IDLE:

```
import sys

while True:
    response = input('Type exit to exit. ')
    if response == 'exit':
        sys.exit()
    print('You typed ' + response + '.')
```

```
You typed __PYTHON_EL_eval("try:\n    with open('c:/Users/BIRKEN~1/AppData/Local/Temp/babel-KmZkAf/python-EJK1Jc') as __org
```

- Run this program in IDLE. It contains an infinite loop with no `break` statement and can only be terminated by either entering 'exit', or by killing the process (closing IDLE).

8. Short program: Guess the Number

- We're going to bring the last few topics together in a complete little game script. The same mechanics will be required for the "Rock, Paper, Scissors" home programming assignment
- This example also demonstrates an exemplary solution path:
  1. Understand what's asked from you (requirements)
  2. Understand what the program needs from you (input)
  3. Understand what's the result supposed to look like (output)

4. Write the process as pseudocode (without syntax)
5. Create a process diagram (with commands)
6. Code the Python program (source code)
7. Run, test and debug the source code
8. Fix pseudocode/diagram accordingly.
9. Identify extensions.
10. Implement extensions (repeat steps 4-8).

- Write a 'Guess the number' game. When you run the program, the output should look like this:

```
Enter number between 1 and 20:
Take a guess: 10
Your guess is too high.
Take a guess: 2
Your guess is too low.
Take a guess: 8
Your guess is too high.
Take a guess: 3
Your guess is too low.
Take a guess: 7
Good job! You guessed my number in 5 guesses!
```

Figure 4: Desired output of guessTheNumber.py

- The program should generate a random number between 1 and 20.
- Enter the source code into the IDLE file editor, or into Colab, and save as `guessTheNumber.py`.
- Solution path/pseudocode (code highlighted)
    1. `import random` module.
    2. Generate a `random` number.
    3. Store number in `num`.
    4. Set `attempt` (number of guesses) to 0.
    5. Get `input` number `guess` from user.
    6. Increase `attempt` by 1
    7. Check if `guess` is the same as `num`
    8. Print success message and `attempt` value
    9. End program
    10. Otherwise, check if `guess` is smaller than `num`
    11. Print information
    12. Otherwise, check if `guess` is larger than `num`
    13. Print information
    14. Return to step 3

- BPMN Process diagram:

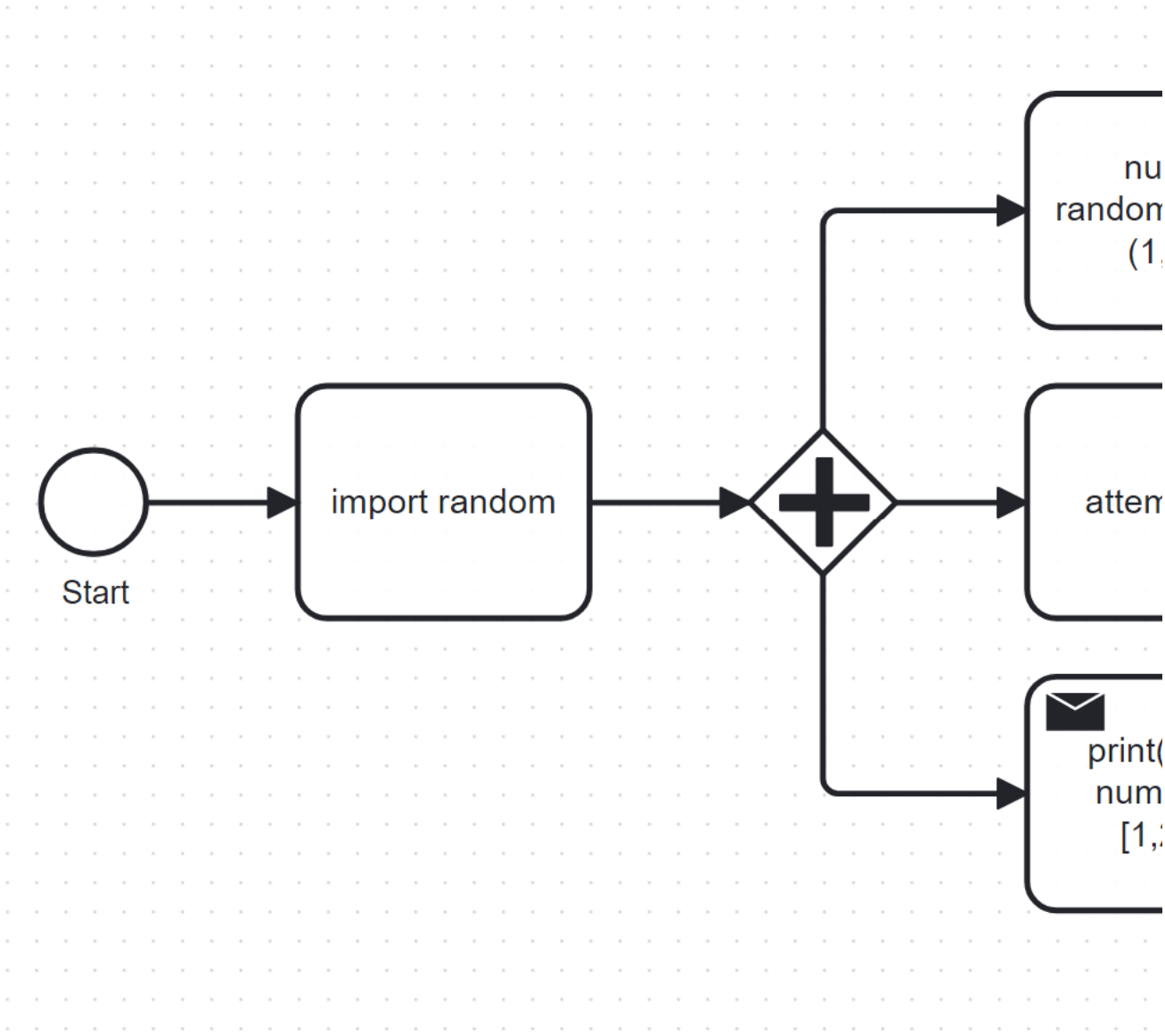

Figure 5: Flow diagram for guessTheNumber.py

- Solution Python code (16 + 5 lines): the `continue` commands are not absolutely necessary for the program to work because at the end of the `if` and `elif` statement, there is no other place to go but the top of the infinite loop.

```
# import random module
import random
# pick random number between 1 and 20
num = random.randint(1,20)
# set attempts counter to 0
attempt = 0
# ask user for number guess
print('Enter number between 1 and 20: ')
# infinite loop until number is guessed
while True:
    guess = int(input('Take a guess: '))
    attempt = attempt + 1
    if guess < num:
        print('Your guess is too low.')
        continue
    elif guess > num:
        print('Your guess is too high.')
        continue
    else:
        print('Good job! You guessed my number in ' + str(attempt) + ' guesses!')
        break
```

```
You typed __PYTHON_EL_eval("try:\n    with open('c:/Users/BIRKEN~1/AppData/Local/Temp/babel-KmZkAf/python-c8Qqjy') as __org
```

- Program extensions:
  1. Make program safe against no/wrong input (exception handling): currently, it terminates with an error if a floating-point number or a letter or nothing is entered by mistake.
  2. Exchange the infinite `while` loop by a `for` loop with a set number of allowed guesses (most games don't go on forever).
- Play the `while` loop version of this program at pythontutor.com
- Visualize a `for` loop version of this program at: pythontutor.com.
- What did you learn?
  1. For best productivity and learning, follow a solution path - don't just "code away"
  2. For best learning effects find different solutions to the same problem.
  3. For best results, handle exceptions. Balance exception handling with usability and performance.
  4. There is always more than one solution, usually many. There is no best solution to a programming problem that satisfies all requirements, even the unspoken ones, equally well.

9. References

- Sweigart, A. (2019). Automate the Boring Stuff with Python. NoStarch. URL: automatetheboringstuff.com

Author: Marcus Birkenkrahe

Created: 2023-06-09 Fri 08:42