Intro to programming 8

Henri Vandendriessche
henri.vandendriessche@ens.fr

2022-10-25

Your computer will do only what you tell it to do; it won't read your mind and do what you intended it to do. Even professional programmers create bugs all the time, so don't feel discouraged if your program has a problem.

Fortunately, there are a few tools and techniques to identify what exactly your code is doing and where it's going wrong. First, you will look at logging and assertions, two features that can help you detect bugs early. In general, the earlier you catch bugs, the easier they will be to fix.

Debugging

The easiest debugging rule

- When your program do what you asked (what you wrote) but not what you wanted. . .

The easiest debugging rule

- When your program do what you asked (what you wrote) but not what you wanted. . .
- The simplest and easiest way to see if there is a problem in you program is to check your variables at every key points of your program:

The easiest debugging rule

- When your program do what you asked (what you wrote) but not what you wanted...
- The simplest and easiest way to see if there is a problem in you program is to check your variables at every key points of your program:
  - When you perform a operation on your variable

The easiest debugging rule

- When your program do what you asked (what you wrote) but not what you wanted...
- The simplest and easiest way to see if there is a problem in you program is to check your variables at every key points of your program:
  - When you perform a operation on your variable
  - At the end of loop

The easiest debugging rule

- When your program do what you asked (what you wrote) but not what you wanted. . .
- The simplest and easiest way to see if there is a problem in you program is to check your variables at every key points of your program:
  - When you perform a operation on your variable
  - At the end of loop
  - At the end of a function if there is a return statement

The easiest debugging rule

- When your program do what you asked (what you wrote) but not what you wanted. . .
- The simplest and easiest way to see if there is a problem in you program is to check your variables at every key points of your program:
    - When you perform a operation on your variable
    - At the end of loop
    - At the end of a function if there is a return statement
    - When you import data from a file

The easiest debugging rule

- When your program do what you asked (what you wrote) but not what you wanted. . .
- The simplest and easiest way to see if there is a problem in you program is to check your variables at every key points of your program:
  - When you perform a operation on your variable
  - At the end of loop
  - At the end of a function if there is a return statement
  - When you import data from a file
- What is the best way to check on your variables and their types ?

The easiest debugging rule

- When your program do what you asked (what you wrote) but not what you wanted...
- The simplest and easiest way to see if there is a problem in you program is to check your variables at every key points of your program:
    - When you perform a operation on your variable
    - At the end of loop
    - At the end of a function if there is a return statement
    - When you import data from a file
- What is the best way to check on your variables and their types ?
    - print it: **print(your_variable)**

The easiest debugging rule

- When your program do what you asked (what you wrote) but not what you wanted. . .
- The simplest and easiest way to see if there is a problem in you program is to check your variables at every key points of your program:
    - When you perform a operation on your variable
    - At the end of loop
    - At the end of a function if there is a return statement
    - When you import data from a file
- What is the best way to check on your variables and their types ?
    - print it: **print(your_variable)**
    - print the type of your variable: **print(type(your_variable))**

Try and except statements 1/4

- If you have an error in your script, the execution is stopped.

Try and except statements 1/4

- If you have an error in your script, the execution is stopped.
- Example: What's wrong in the following script:

```python
def isDivided(divisor):
    return 42 / divisor

print(isDivided(2))
print(isDivided(12))
print(isDivided(0))
print(isDivided(3))
```

Try and except statements 2/4

- If you have an error in your script, the execution is stropped.
- Example: What's wrong in the following script:

```
def isDivided(divisor):
    return 42 / divisor

print(isDivided(2))
```

```
## 21.0
```

```
print(isDivided(12))
```

```
## 3.5
```

```
print(isDivided(0))
```

```
## Error in py_call_impl(callable, dots$args, dots$keywords): ZeroDivisionError:
```

Try and except statements 3/4

- But you can still have your way around this error:
    - **try :**
    - **except ... :**

```
def isDivided(divisor):
    try:
      return 42 / divisor
    except ZeroDivisionError:
      print("What have I done again...")

print(isDivided(2))
```

```
## 21.0
```

```
print(isDivided(12))
```

```
## 3.5
```

```
print(isDivided(0))
```

```
## What have I done again...
## None
```

```
print(isDivided(3))
```

Try and except statements 4/4

- You can as well include the call of your function in the try

```
def isDivided(divisor):
  return 42 / divisor

try:
  print(isDivided(2))
  print(isDivided(12))
  print(isDivided(0))
  print(isDivided(3))

except ZeroDivisionError:
  print("What have I done again...")
```

```
## 21.0
## 3.5
## What have I done again...
```

Try and except statements 4/4

- You can as well include the call of your function in the try

```python
def isDivided(divisor):
  return 42 / divisor

try:
  print(isDivided(2))
  print(isDivided(12))
  print(isDivided(0))
  print(isDivided(3))

except ZeroDivisionError:
  print("What have I done again...")
```

```
## 21.0
## 3.5
## What have I done again...
```

- Once the execution jumps to the code in the except clause, it does not return to the try clause. Instead, it just continues moving down the program as normal.

Raising Exceptions

Getting the Traceback as a String

Python debugger: PDB