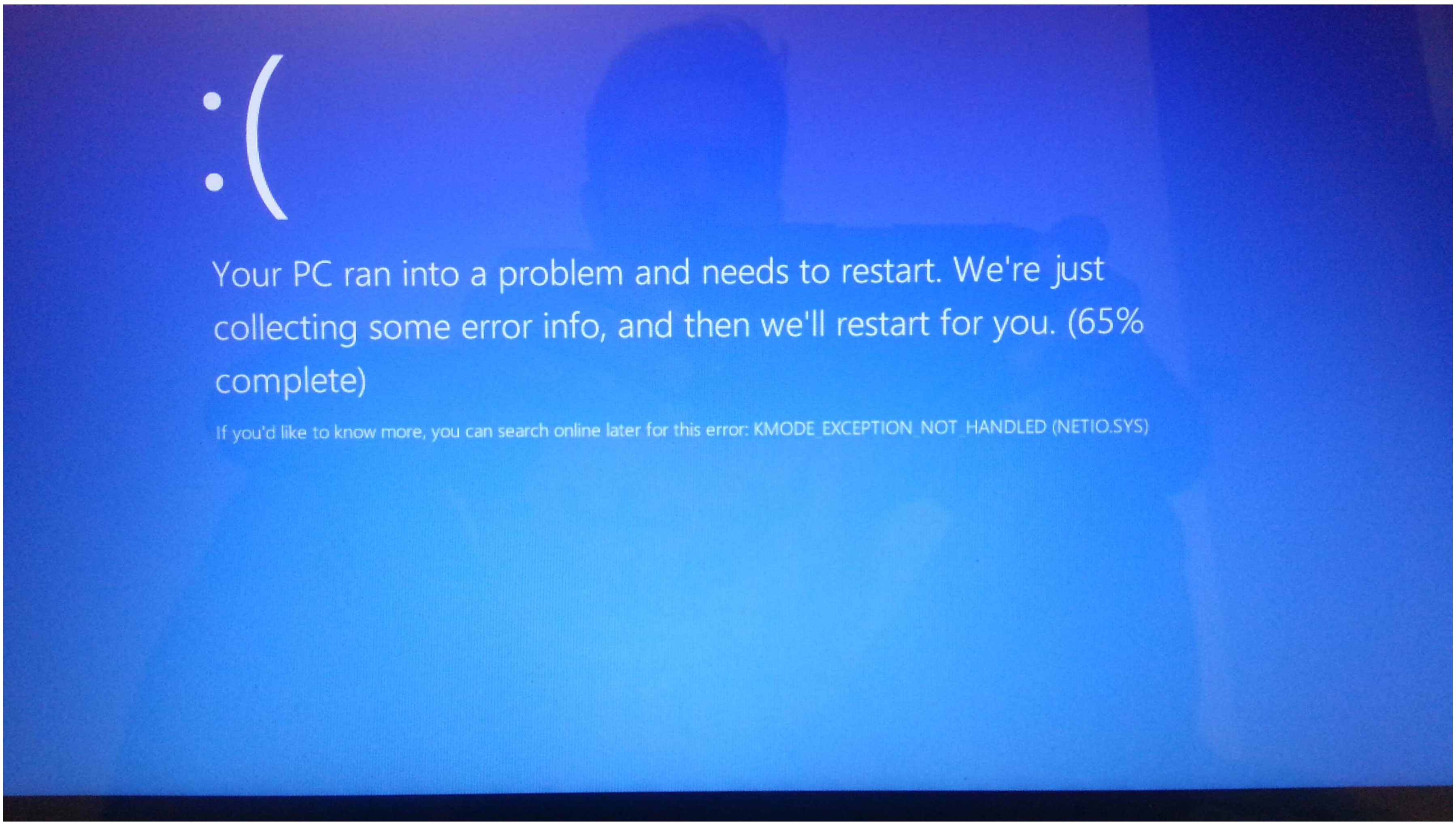
How to break your computer



:(

Your PC ran into a problem and needs to restart. We're just collecting some error info, and then we'll restart for you. (65% complete)

If you'd like to know more, you can search online later for this error: KMODE_EXCEPTION_NOT_HANDLED (NETIO.SYS)

Let's try to break ~~your computer~~ Python

- In Python, you cannot mess up like above
  - Except in exceptional cases
- Don't be afraid to try things out

Let's try to break Python

- Write the following program into Mu

```
print(1 / 0)
```

- Remove the right parenthesis. What do you expect will happen?

- Now write this:

```
    a_list = []
    a_list[1000]
```

Exceptions

Exceptions are system messages that break on a fundamental level. They are typically called something with `Error`:

- `ArithmeticError` (division by 0)
- `IndexError` (list lookup)
- `MemoryError` (out of memory :-/)
- `FileNotFoundError` (you guessed it)
- `ValueError` (when encountering an unexpected value)
- `SyntaxError` (a program that incorrectly written)

They are actually divided into an Exception hierarchy (https://docs.python.org/3.7/library/exceptions.html#exception-hierarchy)

Safely running code

In Python you can **catch** these errors:

```
try:
    print(1 / 0)
except:
    print('Uh-oh, that did not go well')
```

So, what's the difference?

With the `try ... except` your program continues instead of breaking down.

Try to write these two lines by themselves first, and then encapsulate *the first line* inside `try ... except`:

```
print(1 / 0)
print('Don\'t worry about the aftermath')
```

What do you expect will happen? Run it in the debugger.

Breaking your own programs

Exceptions typically exist because the program *encounters something exceptional*. What if you do too?

Type in the following:

```
character = input('Give me a character: ')
if (len(character) == 1):
    print('You wrote ' + character)
else:
    print('Bad human')
```

What do we do in the `else` clause? We asked for a single character, but we got something different.

Raising an exception

If we found something exceptional, we can tell the user by *raising* an exception.

Type in the following:

```
character = input('Give me a character: ')
if (not len(character) == 1):
    raise ValueError

print('You wrote ' + character)
```

In the code we know that *if* the code continues below the if statement, the `character` variable only contains one character.

You can also provide error messages:

```
character = input('Give me a character: ')
if (not len(character) == 1):
    raise ValueError('I expected a character, but received ' + character)

print('You wrote ' + character)
```

Functions and objects

- Functions are like variables, but requires parenthesis
  - `range(0, 10)`

- Objects are like functions, but in upper case
    - `ValueError('I expected ...')`
- Functions perform some *functionality*, while objects describe some *thing*
    - We will talk more about functions on Wednesday and objects on Friday.

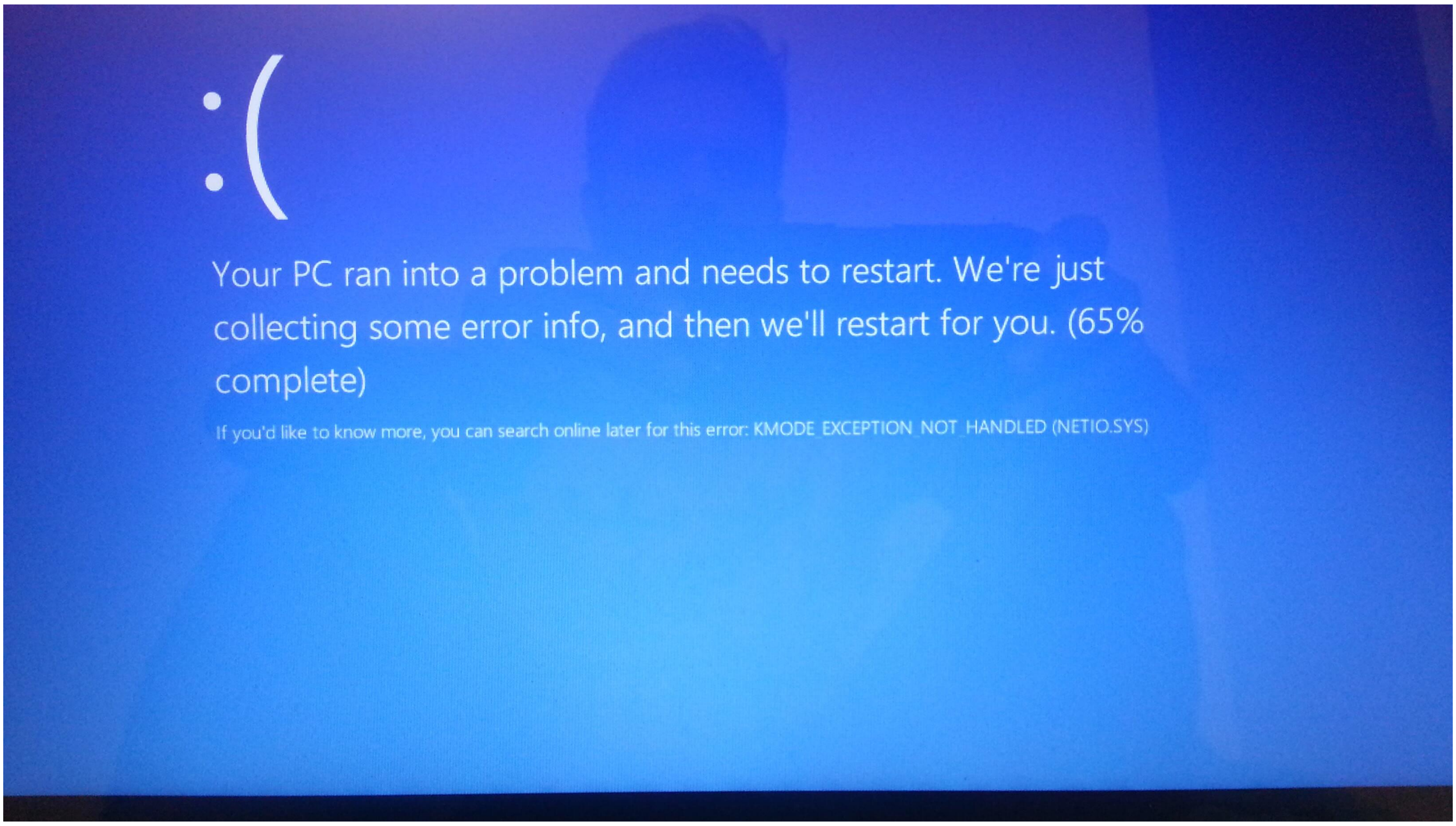
Exercises

- Use the cookie dough program from the second lesson:

```
data = input('How much do you like cookie dough?')
data = int(data)
print('You ' + 'really ' * data + 'like cookie dough')
```

- While running it in the *debugger*, type in a random string instead of a number
    - On which line exactly does it break?
- Wrap the line in a `try ... except`. If the program breaks, default to the number `1`

How to break your computer



:(

Your PC ran into a problem and needs to restart. We're just collecting some error info, and then we'll restart for you. (65% complete)

If you'd like to know more, you can search online later for this error: KMODE_EXCEPTION_NOT_HANDLED (NETIO.SYS)

Let's try to break ~~your computer~~ Python

- In Python, you cannot mess up like above
    - Except in exceptional cases
- Don't be afraid to try things out

Let's try to break Python

- Write the following program into Mu

```
print(1 / 0)
```

- Remove the right parenthesis. What do you expect will happen?

- Now write this:

```
    a_list = []
    a_list[1000]
```

Exceptions

Exceptions are system messages that break on a fundamental level. They are typically called something with `Error`:

- `ArithmeticError` (division by 0)
- `IndexError` (list lookup)
- `MemoryError` (out of memory :-/)
- `FileNotFoundError` (you guessed it)
- `ValueError` (when encountering an unexpected value)
- `SyntaxError` (a program that incorrectly written)

They are actually divided into an Exception hierarchy (https://docs.python.org/3.7/library/exceptions.html#exception-hierarchy)

Safely running code

In Python you can **catch** these errors:

```
try:
    print(1 / 0)
except:
    print('Uh-oh, that did not go well')
```

So, what's the difference?

With the `try ... except` your program continues instead of breaking down.

Try to write these two lines by themselves first, and then encapsulate *the first line* inside `try ... except` :

```
print(1 / 0)
print('Don\'t worry about the aftermath')
```

What do you expect will happen? Run it in the debugger.

Breaking your own programs

Exceptions typically exist because the program *encounters something exceptional*. What if you do too?

Type in the following:

```
character = input('Give me a character: ')
if (len(character) == 1):
    print('You wrote ' + character)
else:
    print('Bad human')
```

What do we do in the `else` clause? We asked for a single character, but we got something different.

Raising an exception

If we found something exceptional, we can tell the user by *raising* an exception.

Type in the following:

```
character = input('Give me a character: ')
if (not len(character) == 1):
    raise ValueError

print('You wrote ' + character)
```

In the code we know that *if* the code continues below the if statement, the `character` variable only contains one character.

You can also provide error messages:

```
character = input('Give me a character: ')
if (not len(character) == 1):
    raise ValueError('I expected a character, but received ' + character)

print('You wrote ' + character)
```

Functions and objects

- Functions are like variables, but requires parenthesis
  - `range(0, 10)`

- Objects are like functions, but in upper case
  - `ValueError('I expected ...')`
- Functions perform some *functionality*, while objects describe some *thing*
  - We will talk more about functions on Wednesday and objects on Friday.

Exercises

- Use the cookie dough program from the second lesson:

```
data = input('How much do you like cookie dough?')
data = int(data)
print('You ' + 'really ' * data + 'like cookie dough')
```

- While running it in the *debugger*, type in a random string instead of a number
  - On which line exactly does it break?
- Wrap the line in a `try ... except`. If the program breaks, default to the number `1`