Writing your first program

- Last session you installed Mu.
- Mu is a **Python code editor**
  - Meaning that it is smarter than just a text editor. You will see why later today.

- Open the Mu editor from the command line via the command `mu-editor`
- Now, write the code from the hand-out into a file in Mu
  - Copy the text **exactly** as it stands
  - When you are done, press run (▶)
- You will likely get an error, and that is completely okay
- Try to find the typo that caused the error and fix it

In [ ]:

```
# My first Python program
name = input('What is your name? ')
print('Hello, ' + name + '. Time to play hangman!')
print('Start guessing...')

secret_word = 'secret'
guesses = ''
turns = 10

while turns > 0:
    failed = 0

    for char in secret_word:
        if char in guesses:
            print(char, end='')
        else:
            print('_', end='')
            failed += 1

    if failed == 0:
        print('\nYou won!')
        break

    print()
    guess = input('guess a character: ')

    if len(guess) > 1:
        print('Only one character at the time, please!')
        continue

    guesses += guess

    if guess not in secret_word:
        turns -= 1
        print('Wrong\n')
        print('You have', turns, 'more guesses')
        if turns == 0:
            print('You lose!')

```

Diving into Python

We'll now start to cover basic Python.

During this lecture, we will ask you to write down code. So have your Mu editor ready. Preferably using an empty code file. You can call it whatever you want, for instance `session2.py`.

Data Input

In Python you can get input from the user via the keyboard with the help of the `input` statement.

```
input('Give me some data: ')
```

Write the above code into Mu and execute it by pressing the ▶ `Run` button.

Wait, what just happened?!

Two things happened:

1. A window appeared called `Running ...` with a the text `Give me some data:` and a blinking cursor!
2. The ▶ `Run` button became a **`x`** `Stop` button!

Now type in some text in the field below and press `Enter` . When you do that `>>>` appears. This means that your application terminated (finished). Press F5 again or the **`x`** `Stop` button to close the window again.

Try to change the text a bit (for instance `Give me MORE data:`) and run the program again!

Did it do what you expected it to do?

Data Output

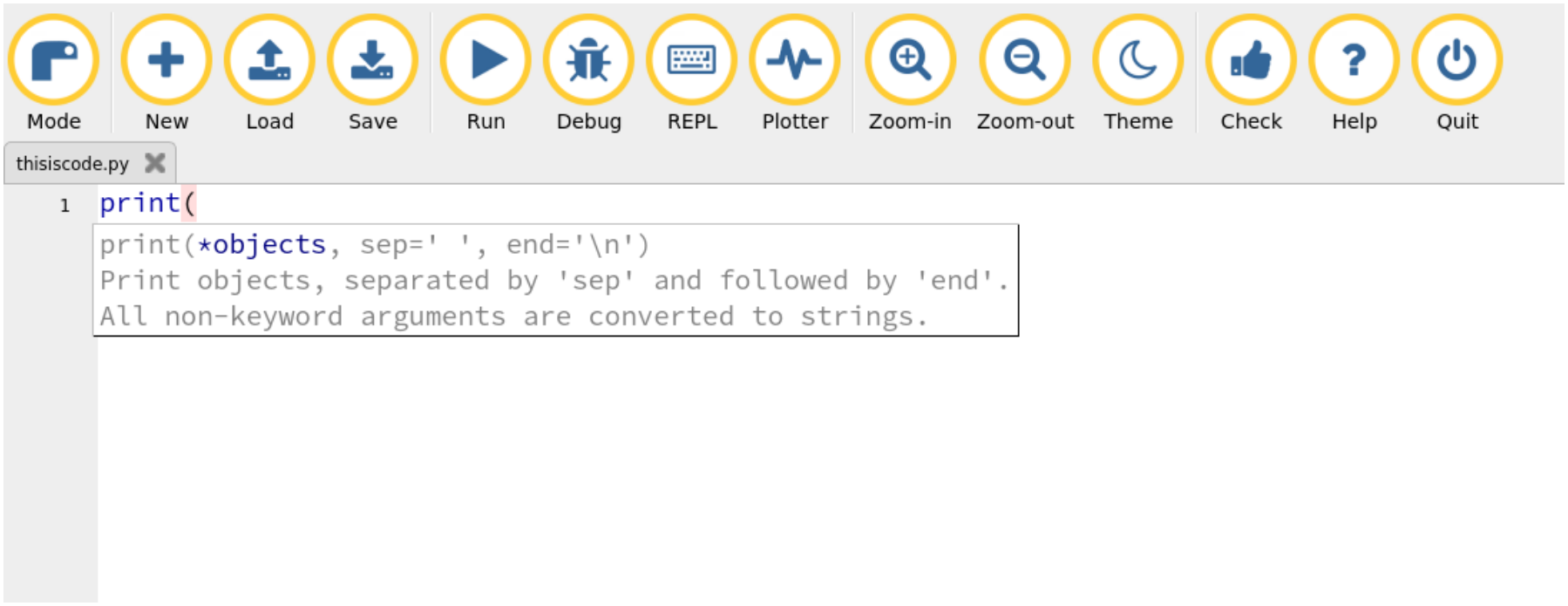
In Python you can display information to the user with a `print` statement.

```
print('Hey, wait! I have to tell you something important.')
```

Delete your previous code and replace it with the code above. Then execute the code by hitting F5 or by clicking ▶ `Run` .

Mu can help you!

Notice that when you write `print(` something happens. This is called documentation. Focus on the first couple of words for now.



A program

> A computer is a device that can perform actions on input (which is also called data). The actions are specified by a program, which is a sequence of instructions.
>
> Interactive Python (http://interactivepython.org/runestone/static/StudentCSP/CSPTuring/whatIsComputer.html)

```
                +---------+
User input  ->  | Program |  ->  Program output
                +---------+        (typically screen)
```

A program

> A computer is a device that can perform actions on input (which is also called data). The actions are specified by a program, which is a sequence of instructions.
>
> Interactive Python (http://interactivepython.org/runestone/static/StudentCSP/CSPTuring/whatIsComputer.html)

```
                +------------+
User input  ->  | Processing |  ->  Program output
                +------------+        (typically screen)
```

Data Processing

Remembering

The first important step in a program is to remember stuff, otherwise we cannot do any kind of processing.

Think about your `input('Give me some data: ')`. The second you pressed enter, that data disappeared. Gone.

Before we can do any processing, we have to save it somewhere. Like putting stuff on a shelf.

Saving into a placeholder

We need a place to store things. Safely, so we can find it again.

That placeholder is called a variable.

Saving with `=`

Our vaults have names to describe what they contain. So a vault called `number` probably contains a number. While a vault called `first_name` probably contains someones first name.

This piece of python code:

```
person = 'Mother Theresa'
```

Is the same as saying:

1. Build me a vault and call it `person`
2. Take the data `'Mother Theresa'` and store it safely inside the vault

Using data

We can now use the data hidden inside the variable `person` like this:

```
print(person)
```

Note that there are no quotes around `person`.

And we can re-use the variable twice or even more!

```
print(person)
print(person)
```

Programs without user input / interaction

Using these vaults, we can easily create programs without user input / interaction. Like this:

```
first_person = 'Albert Einstein'
second_person = 'Stephen Hawkings'

print('EPIC RAP BATTLES OF HISTORY!')
print(first_person)
print('VS.')
print(second_person)
print('BEGIN!')
```

Write down and run this program. Now run it again. And again.

When you run this program, the behaviour will never change. Unless you manually change the variables. Let's do that now so we can change them into our favourite epic rap battles of history characters!

Now that we can store things, we can combine input and output. This is your very first program!

```
data = input('Give me some data: ')
print(data)
```

Type this into Mu. When you're done, verbally explain what is happening step by step with your neighbour.

Variables and data

Notice that we refer to variables differently than we refer to data.

We have to tell Python when we mean "Create a variable for me" and when we mean "Create the letters `Albert Einstein`".

- Variables are written **without** surrounding `'` quotes
- Text is written **with** surrounding `'` quotes

Before we *created* a variable by storing something in it, that we later retrieved:

```
first_person = 'Mother Theresa'
print(first_person)
```

What if we simply try to retrieve it, *without* storing anything in it?

```
print(first_person)
```

What happens when you run that code in Mu?

Compare that to writing:

```
print('first_person')
```

Data types

Variables can contain all kinds of data. And text is just one of them. Anther type of data are numbers.

- This is a piece of text: `'Give me some data: '`
- This is a number: `10`
- This is another number: `1.618`

In Python text is actually called *strings*. *Strings* in Python are surrounded with a single quote like so: `'I am text'`

Note: Python also accepts quotation marks to indicate a string: `"I am text"`. But we will stick to the single quotes.

In Python numbers are divided into *integers* and *floats*. We will cover floats later.

Integers are whole numbers like `1`, `8`, `-15`.

In an empty file in Mu, write an integer. Just an integer. Then run the program.

- What do you expect will happen?
- Did your expectations align with what you saw?

Remember that you can execute your code by hitting F5 or by clicking ▶ `Run`

Why is nothing showing?

Well, you're not outputting anything. Do you remember how to output something in Python?

Now delete the number and write a string instead.

- What do you expect will happen?
- Did your expectations align with what you saw?

Doing stuff with numbers and text

There is a reason for distinguishing between numbers and text. In Mu,

- multiply the integer 4 with 17
- What do you expect will happen?
- Did your expectations align with what you saw?

A multiplication in Python is written with the asterix character (`*`). Multiplying 1 with 2 is written `1 * 2` .

You probably expected that. But what happens if you multiply the string `'ring '` with 8 (don't forget the trailing space)? Try it out.

... Banana phone! Can you guess what happens when you execute the code below?

In [9]:

```
1 print(('Na' * 16) + ' Batman!')
```

```
NaNaNaNaNaNaNaNaNaNaNaNaNaNaNaNa Batman!
```

Oh oh, he just ran code on the screen!

Yes, yes I did. And that's completely fine. This is also Python. The same kind of Python that you write in Mu.

The difference is that it automatically prints the output onto these presentation slides.

Can you store the value `'D' + 'K'` in the variable called `text` ?

In [12]:

```
print('D' + 'K')
```

```
DK
```

Can you guess what the variable contains? Try to print the content of the variable and see if it matches your expectation.

In [14]:

```
text = 'D' + 'K'
print(text)
print(text)
```

```
DK
DK
```

Variables are great because they help us remember things. Your variable `text` now exists until you either:

- kill the application that contains the variable (Mu) or
- overwrite the variable

We don't want to close the Mu editor just yet (although we encourage you to try and see what happens), so let's try to overwrite your variable. Store `'Hello'` in the variable `text` and print the content:

In [ ]:

```

```

This form of overwriting *completely removes* the previous value of `text`. We cannot acces the previous data (`'DK'`). You can think of variables as placeholders: now `text` is equivalent to `'Hello'`, so every time we need the string `'Hello'` we can simply write `text`.

This is pretty daft of course because `'Hello'` is not that much more difficult to write than `text`. But you can save much more data in a variable, and then this form of substitution becomes very handy.

Code is executed stepwise

Code follows a strict logic. It executes code line-wise **one line at the time**! No exceptions.

What is the result of this?

In [15]:

```
text = 'Hello'
text = 'World'
print(text)
```

```
World
```

In [16]:

```
number = 10
number = number + 1
print(number)
```

```
11
```

Converting strings to integers

A Python program treats all input that it receives from the keyboard as a string. That is, as a sequence of textual characters. Even when you enter numbers, such as `1`, `2`, `3`, etc.

If you wanted to input numbers and treat them as numbers and not as strings, you have to tell your program to convert the data type of your input into an integer (`int`).

```
data = '10'
number = int(data)
print(number)
```

Type in this program and explain to your neighbour what it does:

```
data = input('Give me a positive number: ')
print('I say "Hello" multiple times: ' + data)
data = int(data)
print(data * 'Hello')
```

Describe the difference between the two programs:

```
data = input('Give me a positive number: ')
print('I say "Hello" multiple times: ' + data)
data = int(data)
print(data * 'Hello')
```

```
data = input('Give me a positive number: ')
print('I say "Hello" multiple times: ' + data)
print(int(data) * 'Hello')
```

In [ ]:

```

```

Sometimes Python gets confused about the type of your data. We saw that you could multiply a `string` with an `integer`. But what about `adding` (+) a string with an `integer` ? Try to execute the following:

In [ ]:

```
1 'Space' + 10
```

What you just saw was an `Error`. You will see a lot of these in your programming career. But don't worry, it helps us to understand the root of the problem.

```
TypeError: must be str, not int
```

In other words Python gets confused about the *types* of your data. Python sees an `int` (`10`), but is expecting a `str` (`string`). Luckily we can fix this. Just like we could take a `string` and turn it into an `int` (using `int`), we can also take an `integer` and turn it into a `str` (`string`). Can you guess how and fix the code below?

In [ ]:

```
1 'Space' + 10
```

Operations

So far you have worked with `+`, `*` to manipulate your data. These are called `operators` because they `operate` on your data.

The following piece of code uses the operator `*`. But it returns an empty string (`''`). Can you figure out why? And can you make it return `'Hello'`?

In [ ]:

```
1 'Hello' * 0
```

Which of these are operators and which are data?

- `+`
- `8`
- `'Yolo'`
- `*`
- `'C4rp3 Di3m'`
- `/`
- `'*'`

Before you saw an example where you added `strings` and `integers`. Let's try to expand that. Can you make the below code return '7ate9'?

In [ ]:

```
1 7 + 8 + 9
```

A cookie dough program

Now that we have learned about getting input (with `input`), printing text (strings), data types (`str` and `int`) and operators (`+`, `*` and `/`), we'll put it all together.

Here is the requirement for your program: Your program will identify how much the user likes cookie dough. The user can give a number between 1 and 10 to indicate just how much he/she likes it. If the user types 1, you should print `'I really like cookie dough'`. If the user types 3, you should print `'I really really really like cookie dough'`. If the user types 10, we have a serious cookie dough lover, so you should print `'really'` 10 times.

You can assume that the user will never enter a number outside the range of 1-10.

In [ ]:

```
1
```

Hint: Developing applications is hard. The best approach is to break the problem down into steps. What is the first thing you need for your program?

Hint: The first thing you need is input. Look above: do you remember how to get some input from the user? Now you can proceed: what type is the input you just received? And what type do you actually need?

Hint: Now that you have a number from the user (between 1 and 10) the second step is to use that number. For what do you have to use the number?

Hint: The number determines how many times the string `'really'` should be printed. Do you remember what operator can print a string multiple times?

Congratulations!

You have now written your first program. And it's actually a pretty useful program. Think about it; you have

1. received input from a user
2. processed that into the number you needed
3. used that number to produce some output
4. ..that you printed back to the user

Good job! Next you'll write a more sophisticated program that can talk back to you! Then we will look at exactly why you are writing code like this and why it's so useful.

What is programming actually?

Recently on Twitter, "Describe programming in only six words":
https://twitter.com/hashtag/ProgrammingIn6Words?src=hash
(https://twitter.com/hashtag/ProgrammingIn6Words?src=hash)

Crafting instructions to do important things. @JeffDean

Build. Deploy. Test. Debug. Debug. Debug. @StackOverflow

It's a feature, not a bug @leslieasheppard

Making complex things appear simple. @Grady_Booch

Being forced to think very clearly @erikbryn

Try it. Get feedback. Learn. Repeat. @PragmaticAndy

Writing code to write less code @alhuelamo

Turning confused computers into helpful friends @ossia

The computer cooks. I write recipes. @marcorobotics

It ran on my machine yesterday @unrahu1

Why is this still not working?! @ASpittel

Did you remember to clear your cache? @aburke626

Oh WOW, it's working — but how? @codeorg

Who wrote this garbage? Oh, me. @Fobwashed

Hello, world. All are welcome here. @megahoch

In [17]:

```
1 from IPython.display import YouTubeVideo
2 YouTubeVideo('5S-CREfOnXw')
3
```

Out[17]: