

Data Types, Variables and Expressions

Prof. Harald Gall,

University of Zurich, Department of Informatics

Outline

- Data types
- Variables
- Operators
- Expressions
- Strings
- Syntax and semantics
- Comments
- Example exercises

Data types

- Programs mainly manipulate **data (values or objects)**.
- Each values has a specific type which defines how to interpret the value and what operations (**operators**) can be performed on it

```
>>> print(4)
```

```
4
```

```
>>> print('Hello, World!')
```

```
Hello, World!
```

Scalar Value Types

- *hold a single value*
- Numeric types: **int** (3, 5, etc.), **float** (3.5, 123.23), **complex** (2 + 3j)
- NoneType: **None** (represents the absence of a value)
- Boolean types: **True**, **False**

Compound Value Types

- *hold multiple independent values*
- **Strings:** 'Hello, World!', "Hello, World!"
- **Lists, dictionaries, tuples** (more on that later)
- **User defined types** (more on that later)

Numeric types – Examples

```
>>> print(4)
```

```
4
```

```
>>> print(4.5)
```

```
4.5
```

```
>>> print(2 + 3j)
```

```
(2 + 3j)
```

Numeric types – Checking the type

```
>>> type(4)
<class 'int'>
>>> type(4.5)
<class 'float'>
>>> type(2 + 3j)
<class 'complex'>
```

Numeric types – Operators

Operation	Result
$x + y$	sum of x and y
$x - y$	difference of x and y
$x * y$	product of x and y
x / y	float division of x and y
$x // y$	integer division of x and y
$x \% y$	remainder of x / y (a.k.a. "modulo")
$-x$	x negated
$+x$	x unchanged
<code>int(x)</code>	x converted to integer – watch out, this just prunes the decimal component: <code>int(-3.9)</code> is -3
<code>long(x)</code>	x converted to long integer – Python3 doesn't have long type, Python2 only
<code>float(x)</code>	x converted to floating point
<code>complex(re, im)</code>	a complex number with real part re, imaginary part im(defaults to zero)
$x ** y$ or <code>pow(x, y)</code>	x to the power of y

Numeric Operators – Examples

```
>>> 3 + 5
```

```
8
```

```
>>> 3 / 2
```

```
1.5
```

```
>>> 3 // 2
```

```
1
```

```
>>> 2 ** 3
```

```
8
```

Boolean type – Operators

Operation	Processing	Result
x and y	x is evaluated and only if x is True, y is also evaluated. If x is False, y is never evaluated.	True if x and y are both True
x or y	x is evaluated and only if x is False, y is also evaluated. If x is True, y is never evaluated.	True if x or y or both are True
not x	x is evaluated	True if x is false



Boolean type – Operators

Operation	Processing	Result	
x and y	x is evaluated and only if x is True, y is also evaluated. If x is False, y is never evaluated.	True if x and y are both True	<pre>>>> True and True True >>> True and False False >>> False and True False >>> True or False True >>> False or True True >>> not True False >>> not False True</pre>
x or y	x is evaluated and only if x is False, y is also evaluated. If x is True, y is never evaluated.	True if x or y or both are True	
not x	x is evaluated	True if x is false	

Boolean type – Operators

Operation	Processing	Result		
x and y	x is evaluated and only if x is True, y is also evaluated. If x is False, y is never evaluated.	True if x and y are both True	>>> True and True True >>> True and False False >>> False and True False >>> True or False True >>> False or True True >>> not True False >>> not False True	>>> def return_true(): ... print("True function") ... return True ... >>> def return_false(): ... print("False function") ... return False ... >>> return_true() or return_false() True function True # return_false() is not evaluated! >>> return_false() and return_true() False function False # return_true() is not evaluated!
x or y	x is evaluated and only if x is False, y is also evaluated. If x is True, y is never evaluated.	True if x or y or both are True		
not x	x is evaluated	True if x is false		

Variables

- A variable binds a name to a specific value through the assignment operation

```
>>> message = 'Hello, World!'
```

```
>>> n = 100
```

```
>>> b = True
```

Variables

- the variable name is then used to refer to the stored value

```
>>> message = 'Hello, World!'
```

```
>>> print(message)
```

```
Hello, World!
```

Variables

- a variable name can be re-assigned

```
>>> message = 'Hello, World!'
```

```
>>> print(message)
```

```
Hello, World!
```

```
>>> message = "Hello, Bob!"
```

```
>>> print(message)
```

```
Hello, Bob!
```

Variable names

- Can contain characters, numbers and '_'
- Case sensitive: **Banana** is not the same as **banana**
- Choose meaningful names
- Naming convention: lowercase with words separated by underscores
- Cannot use reserved keywords as variable names

Reserved keywords

and	assert	break	class	continue	def
del	elif	else	except	exec	
finally	for	from	global	if	import
in	is	lambda	not	or	pass
print	raise	return	try	while	yield

Expressions

- We can combine values, variables and operators to build complex expressions that are evaluated by the interpreter and produce a result

```
>>> x = 2
>>> y = 3
>>> z = 4
>>> x + y * z + 100
114
```

How are expressions evaluated?

The **operator precedence** determines the order in which operators are evaluated. Operators are evaluated in the following order:

- Parentheses – ()
 - Exponentiation - **
 - Multiplication and Division - *, /, //
 - Addition and Subtraction - +, -
-
- Operators with the same precedence are evaluated from left to right

Operators precedence – Example

$(2 * 3 - 1) ** 2 + 1$

Operators precedence – Example

$(2 * 3 - 1) ** 2 + 1$

$(2 * 3 - 1) ** 2 + 1$

Operators precedence – Example

(2 * 3 - 1) ** 2 + 1

(2 * 3 - 1) ** 2 + 1

(2 * 3 - 1) ** 2 + 1

Operators precedence – Example

$(2 * 3 - 1) ** 2 + 1$

$(2 * 3 - 1) ** 2 + 1$

$(2 * 3 - 1) ** 2 + 1$

$(6 - 1) ** 2 + 1$

Operators precedence – Example

$(2 * 3 - 1) ** 2 + 1$

$(2 * 3 - 1) ** 2 + 1$

$(2 * 3 - 1) ** 2 + 1$

$(6 - 1) ** 2 + 1$

Operators precedence – Example

$(2 * 3 - 1) ** 2 + 1$

$(2 * 3 - 1) ** 2 + 1$

$(2 * 3 - 1) ** 2 + 1$

$(6 - 1) ** 2 + 1$

$5 ** 2 + 1$

Operators precedence – Example

$(2 * 3 - 1) ** 2 + 1$

$(2 * 3 - 1) ** 2 + 1$

$(2 * 3 - 1) ** 2 + 1$

$(6 - 1) ** 2 + 1$

$5 ** 2 + 1$

Operators precedence – Example

$(2 * 3 - 1) ** 2 + 1$

$(2 * 3 - 1) ** 2 + 1$

$(2 * 3 - 1) ** 2 + 1$

$(6 - 1) ** 2 + 1$

$5 ** 2 + 1$

$25 + 1$

Operators precedence – Example

$(2 * 3 - 1) ** 2 + 1$

$(2 * 3 - 1) ** 2 + 1$

$(2 * 3 - 1) ** 2 + 1$

$(6 - 1) ** 2 + 1$

$5 ** 2 + 1$

$25 + 1$

Operators precedence – Example

$(2 * 3 - 1) ** 2 + 1$

$(2 * 3 - 1) ** 2 + 1$

$(2 * 3 - 1) ** 2 + 1$

$(6 - 1) ** 2 + 1$

$5 ** 2 + 1$

26

String

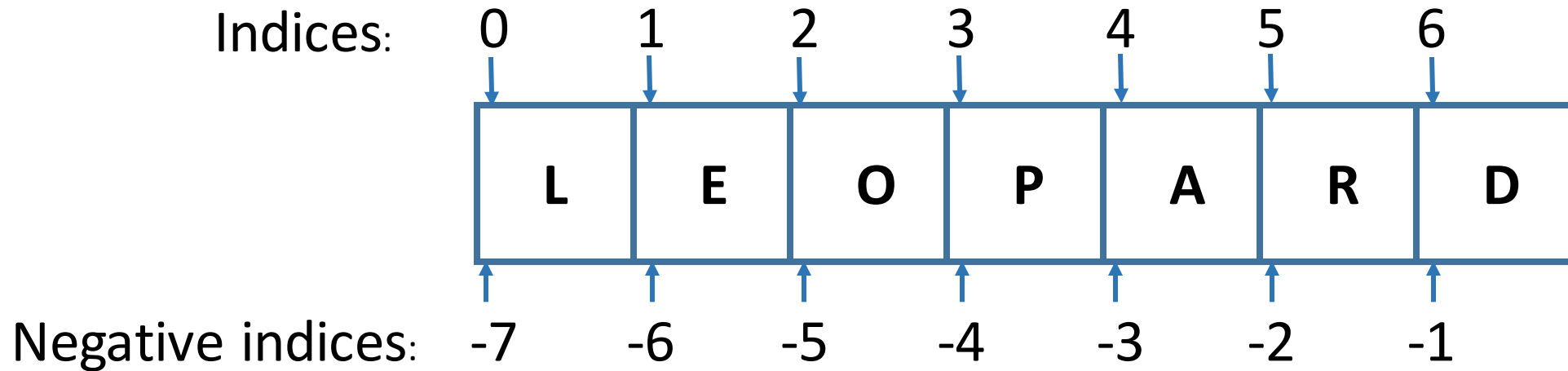
- Is a compound data type (made up of a sequence of characters)

```
>>> animal = 'dog'
>>> print(animal)
dog
>>> animal = "cat"
>>> print(animal)
cat
```

Strings – Accessing the characters

```
>>> animal = 'leopard'
>>> animal[0]
l
>>> animal[1]
e
>>> animal[-1]
d
>>> animal[10]
IndexError: string index out of range
```

Strings – Accessing the characters



String slicing

```
>>> animal = 'leopard'
>>> animal[0:3]
'leo'
>>> animal[:3]
'leo'
>>> animal[5:7]
'rd'
>>> animal[5:]
'rd'
```

String operators

Operation	Result
$x + y$	concatenation
$x * n$	x is a string, n is a number, repeats string x n times

String operators – Examples

```
>>> firstname = 'Alice'
>>> lastname = 'Smith'
>>> firstname + lastname
'AliceSmith'
>>> firstname + ' ' + lastname
'Alice Smith'
>>> firstname * 3
'AliceAliceAlice'
>>> 3 * lastname
'SmithSmithSmith'
```

String formatting

- You can build complex string expression using the concatenation operator (+), but this can become cumbersome
- A better way is to use "String formatting" or "String interpolation":
 - "Old style" string formatting: `"Hello, %!" % x`
 - "New style" string formatting: `"Hello, {}".format(x)`
 - String interpolation: `f"Hello, {x}!"`
- From Python 3.6, string interpolation (a.k.a. "f-strings") are the easiest method.

String interpolation – Examples

```
>>> name = 'Alice'
>>> f'Hello, {name}!'
'Hello, Alice!'
>>> age = 25
>>> f'Hello, {name}! You are {age} years old.'
'Hello, Alice! You are 25 years old.'
```

Syntax & Syntax Errors

- the **syntax** of a programming language is the set of rules that define what combination of symbols (values, variables, expression, operators, etc.) are considered valid
- if you do not follow the rules the interpreter will signal a **syntax error**

```
>>> 3 + 2 +
```

```
File "<stdin>", line 1
```

```
3 + 2 +
```

```
^
```

```
SyntaxError: invalid syntax
```

```
>>> print("Hello)
```

```
File "<stdin>", line 1
```

```
print("Hello)
```

```
^
```

```
SyntaxError: EOL while scanning string literal
```

Other programmatic errors

- Even **syntactically correct** code can be **invalid**, because of improper use of data types or other API-related functionality:

```
>>> 'Hello' + 3
```

```
TypeError: Can't convert 'int' object to str implicitly
```

```
>>> print(name)
```

```
NameError: name 'name' is not defined
```

Semantics

- semantics is the meaning associated with a syntactically correct string of symbols without syntactical errors
- Semantic errors cannot be detected by the interpreter!!

```
>>> x = 10
>>> y = 3
>>> sum = x - y
>>> sum
7
```


Types of errors

- syntactical errors (always signaled by the interpreter)
- other programmatic errors (often signaled by the interpreter)
- semantic error (can only be checked by humans or testing)
 - different meaning than what the programmer actually intended (the program crashes, runs forever or returns a different result than what was actually intended)

Getting data from the user

- to get data from the user you can use the `input` function
- it returns a string containing what the user typed in the terminal

```
>>> name = input('Your name: ')\nYour name:  Bob\n>>> print('Hello, %s!' % name)\n'Hello, Bob!'
```

Comments

- comments are explanations or annotations of source code written by the programmer
- they are generally ignored by the interpreter (or compiler) and should make the code easier to understand
- In Python, anything after an octothorpe (#) is ignored

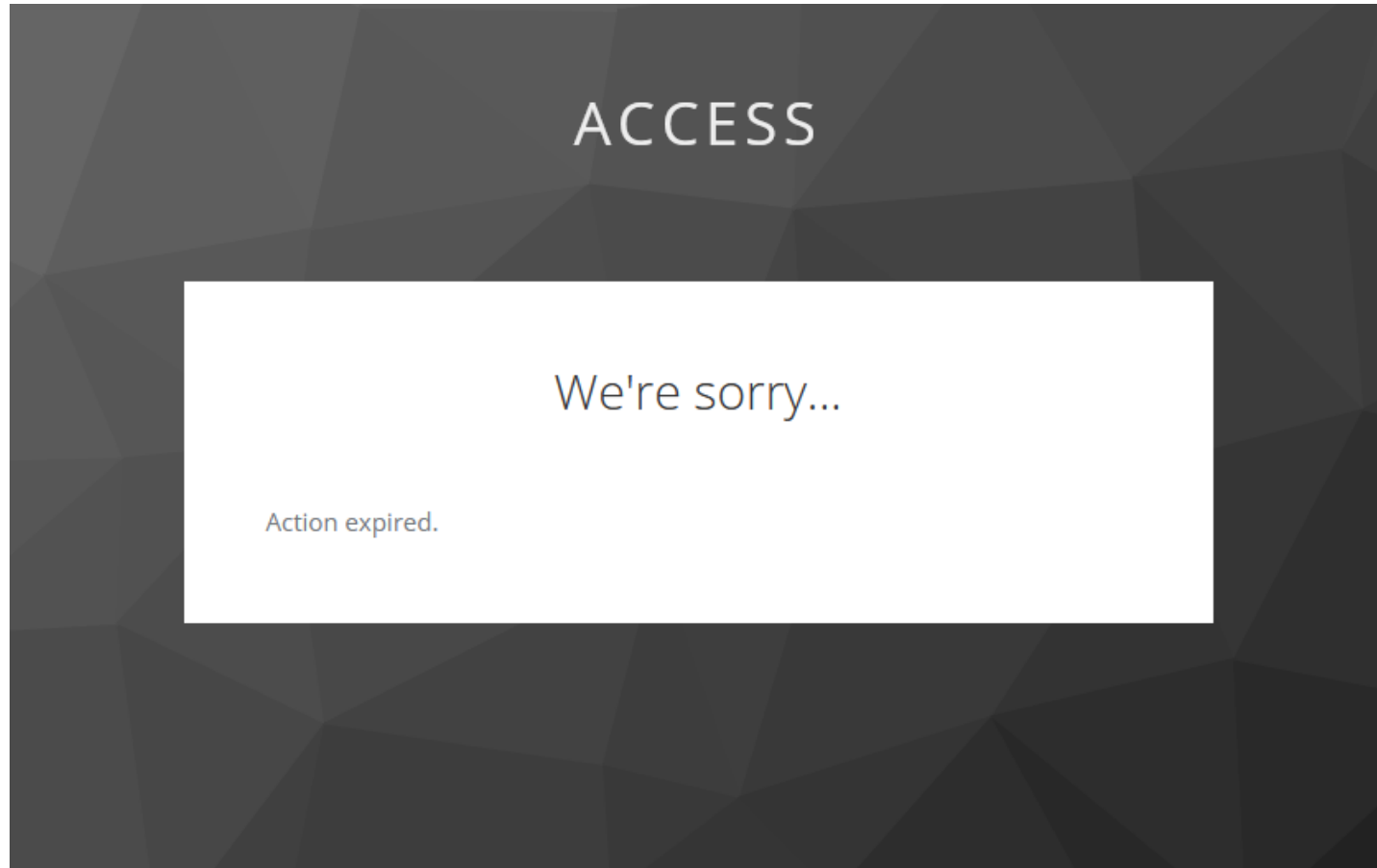
```
# ask for the user name
username = input('Your name: ')
msg = 'Hello, ' + username + '!' # Construct message
print(msg)
```

What we learned today

- There exist different data types (int, float, string, compound...)
- We can assign values to variables
- Basic operators (including %, //, and, and or...)
- Small, self-contained code snippets are called "expressions"
- How to create and slice Strings using indices
- Errors can occur in syntax, invalid use of the language, and semantics
- How to comment in source code

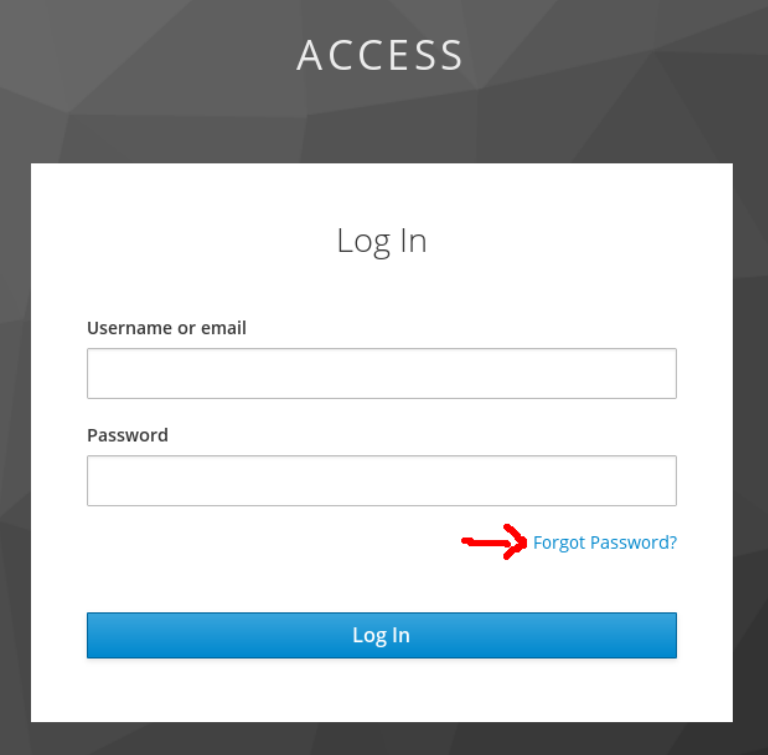
Now, for some more examples...

Addressing some issues...



Accessing ACCESS

- If you haven't booked the Info1 module, you're not added yet
 - <https://www.students.uzh.ch/en/booking.html>
- If you think you did not receive an invite...
 - Double-check your inbox, spam etc. in your webmail at <https://webmail.uzh.ch/>
 - Disable any email forwarding if you enabled it
 - Click the "Forgot Password?" link and put in your **UZH Email address (the same you see in your OLAT profile!)**. You should receive an email within a few minutes.
 - If you do not, please write us at info1@lists.ifi.uzh.ch.



The screenshot shows the ACCESS login interface. At the top, the word "ACCESS" is displayed in a light blue font. Below it, the text "Log In" is centered. There are two input fields: "Username or email" and "Password". To the right of the password field, there is a red arrow pointing to a link that says "Forgot Password?". At the bottom, there is a blue button labeled "Log In".

System Shell (Terminal) vs. Python Shell

```
Command Prompt
(c) 2015 Microsoft Corporation. All rights reserved.

C:\Users\win10>print("hello")
Unable to initialize device PRN

C:\Users\win10>echo "hello"
"hello"

C:\Users\win10>
```

≠

```
Command Prompt - python
Microsoft Windows [Version 10.0.10586]
(c) 2015 Microsoft Corporation. All rights reserved.

C:\Users\win10>python
Python 3.9.7 (tags/v3.9.7:1016ef3, Aug 30 2021, 20:19:38)
Type "help", "copyright", "credits" or "license" for more
>>> print("Hello")
Hello
>>> echo "Hello"
File "<stdin>", line 1
    echo "Hello"
    ^
SyntaxError: invalid syntax
>>> _
```

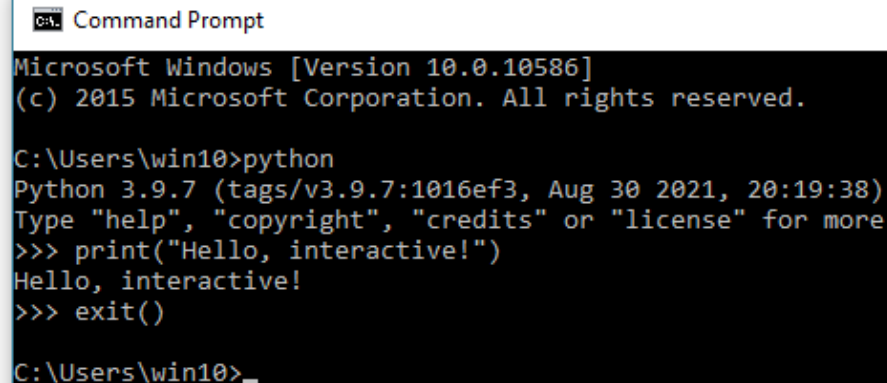
```
shapeshifter@blackrabbit> print("Hello")
zsh: unknown file attribute: H
[1] shapeshifter@blackrabbit> echo "Hello"
Hello
shapeshifter@blackrabbit> 
```

```
shapeshifter@blackrabbit> python3
Python 3.9.7 (default, Aug 31 2021,
[GCC 11.1.0] on linux
Type "help", "copyright", "credits"
>>> print("Hello")
Hello
>>> echo "Hello"
File "<stdin>", line 1
    echo "Hello"
    ^
SyntaxError: invalid syntax
>>> 
```

"Interactive" python session vs. script

- Typing `python`: Starts the python interpreter and puts you into an interactive Python session (`>>>`). Use `exit()` to stop Python.

```
C:\Users\John> python
```



```
Command Prompt
Microsoft Windows [Version 10.0.10586]
(c) 2015 Microsoft Corporation. All rights reserved.

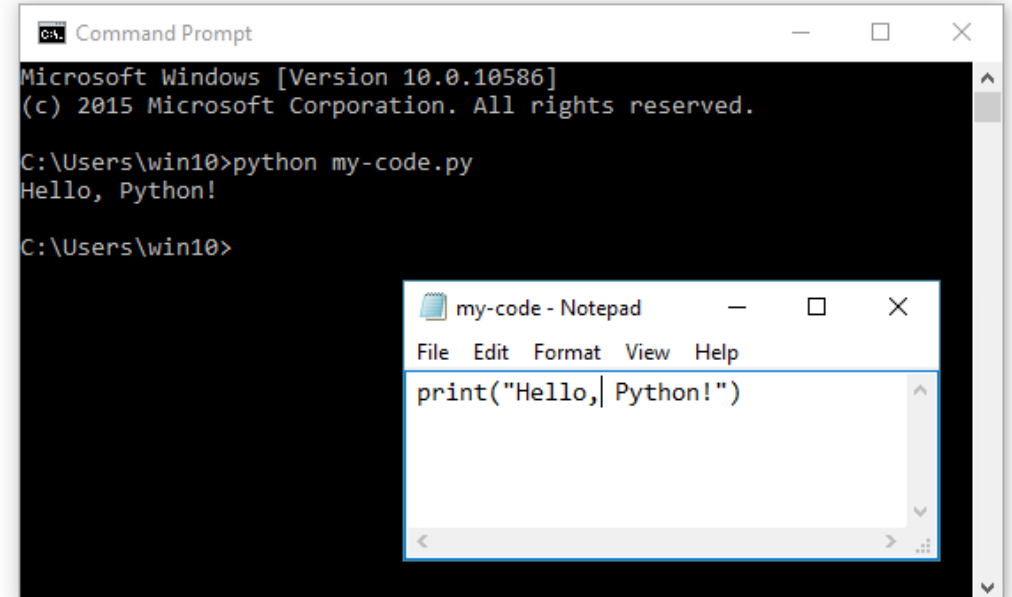
C:\Users\win10>python
Python 3.9.7 (tags/v3.9.7:1016ef3, Aug 30 2021, 20:19:38)
Type "help", "copyright", "credits" or "license" for more
>>> print("Hello, interactive!")
Hello, interactive!
>>> exit()

C:\Users\win10>_
```

Use this to try small snippets!

- Typing `python some.py`: Runs the code contained in some.py using Python. *When the script ends, you're back in your system's terminal.*

```
C:\Users\John> python some-script.py
```



```
Command Prompt
Microsoft Windows [Version 10.0.10586]
(c) 2015 Microsoft Corporation. All rights reserved.

C:\Users\win10>python my-code.py
Hello, Python!

C:\Users\win10>
```

my-code - Notepad

```
File Edit Format View Help
print("Hello, Python!")
```

≠

Data Types, Variables and Expressions

--- Examples

Dr. Carol Alexandru-Funakoshi,
University of Zurich, Department of Informatics

Example 1 – Maths

Using Python as a calculator - compute the following expressions with Python:

- $2 + 2 + 5 \times 10^2$
- $(1 + 10^3) \times (2 + 5^2)$
- $\sqrt[2]{(1 + 10)} - (1 + 10)^3$

Example 1 - $2 + 2 + 5 \times 10^2$

```
>>> 2 + 2 + 5 * 10 ** 2
```

```
504
```

Example 1 - $(1 + 10^3) \times (2 + 5^2)$

```
>>> (1 + 10 ** 3) * (2 + 5 ** 2)  
27027
```

Example 1 - $\sqrt[2]{(1 + 10)} - (1 + 10)^3$

```
>>> (1 + 10) ** (1/2) - (1 + 10) ** 3  
-1327.6833752096445
```

Example 2 – Input and string interpolation

Ask the user the following information: name, age, occupation and print the following message using the provided values:

Hi *NAME*! I see you are *AGE* years old and work as a *OCCUPATION*.

(This is an *interactive* application)

Ask for the name

```
input('What is your name? ')
```

Store what the user has typed in a variable

```
name = input('What is your name? ')
```


Build the message – using + operator

```
name = input('What is your name? ')
age = input('How old are you? ')
occupation = input('What is your occupation? ')
msg = 'Hi ' + name + '! I see you are ' + age + '
old and work as a ' + occupation + '.'
print(msg)
```

Build the message – string interpolation

```
name = input('What is your name?')  
age = input('How old are you?')  
occupation = input('What is your occupation?')  
print(f'Hi {name}! I see you are {age} years old and  
work as a {occupation}.')
```

Example 3 – Modulo & Integer division

Vending machine change:

1. Ask the user for an amount between 1 cent and 99 cents.
2. Print out a combination of coins equal to that amount
 - quarters – 25 cents
 - dime – 10 cents
 - nickel – 5 cents
 - pennies – 1 cent

(This is also known as a "bin packing problem")

Ask for the amount

```
amount = int(input('enter cents value (1-99): '))
```

`int()` is a function to "*cast*" a string to an integer number:

```
>>> "23" + 3
```

```
Traceback (most recent call last):
```

```
  File "<stdin>", line 1, in <module>
```

```
TypeError: can only concatenate str (not "int") to str
```

```
>>> int("23") + 3
```

```
26
```

Print change

```
amount = int(input('enter cents value (1-99): '))
quarters = amount // 25
rest_amount = amount % 25
dimes = rest_amount // 10
rest_amount %= 10
nickels = rest_amount // 5
rest_amount %= 5
pennies = rest_amount
print('%d quarters %d dimes %d nickels %d pennies' %
      (quarters, dimes, nickels, pennies))
```

Example 4 – String functions

Transform the string "Hello, new world!" to the string "Oh, hell no!!!"
(without redefining any letters!)

<https://docs.python.org/3.8/library/stdtypes.html#string-methods>

Indices, ranges, lower/upper and find!

```
s = "Hello, new world!"
space = s[6]                    # " "
exclam = s[-1]                 # "!"
res = (s[4].upper() +          # "O"
       s[0].lower() + s[5] + space + # "h, "
       s[0:4].lower() + space +    # "hell "
       s[s.find('n')] + s[4] +     # "no"
       exclam * 3                 # "!!!"
       )
print(res)
```

Final words

- Use the forgotten password link if you haven't got ACCESS
 - Check your UZH webmail
 - Use "Forgot Password" and enter your UZH email address
 - If that doesn't work, write to info1@lists.ifi.uzh.ch
- Make sure your local Python 3 Installation is working well
 - Also install a good text editor or IDE
- Make sure you know how to navigate the CLI
- Do the exercises, they are probably the most important part of this course
- Don't give up!