Elements of Programming

Winter 2022/23

Lecture Content

Elements of Programming	Fundamental Algorithms	Data Structures	Tree Structures	Graph Structures
Input/Output	*Searching*	*OOP*	*Binary Trees*	*Graphs*
Variables and Data Types	*Sorting*	*Stack / Queue*	*Binary Search Trees*	*Traversal*
Control Structures		*Lists*	*Balanced Search Trees*	*Special Algorithms*
Functions				

Applications

Your first program

- *Python Programming language* (short: Python)
  - Developed in 1991 by Guido van Rossum
  - Current version: 3.10.5 (6.6.2022)
- Programming
  - Usually something you do in a text editor
  - NotePad++ is great, free and sufficient for everything in this course
- Running a program
  - Usually something you do by double clicking an icon
  - Here: Use the command line

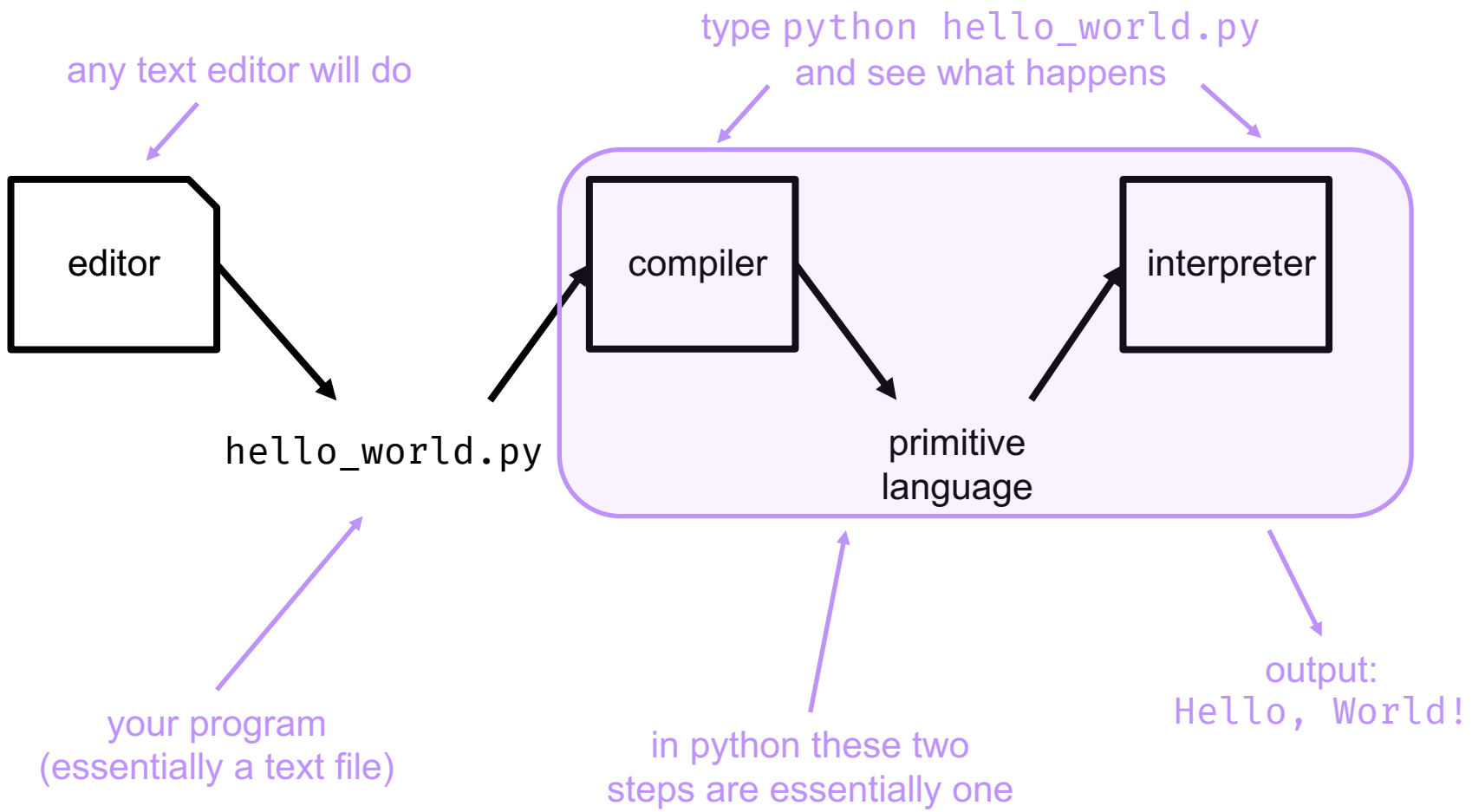
hello_world.py

```
1 # prints "Hello, World!" to standard output
2 print("Hello, World!")
```

- First Line: A comment
  - Interpreter ignores this
- Second line
  - A statement that calls the print function with a single parameter

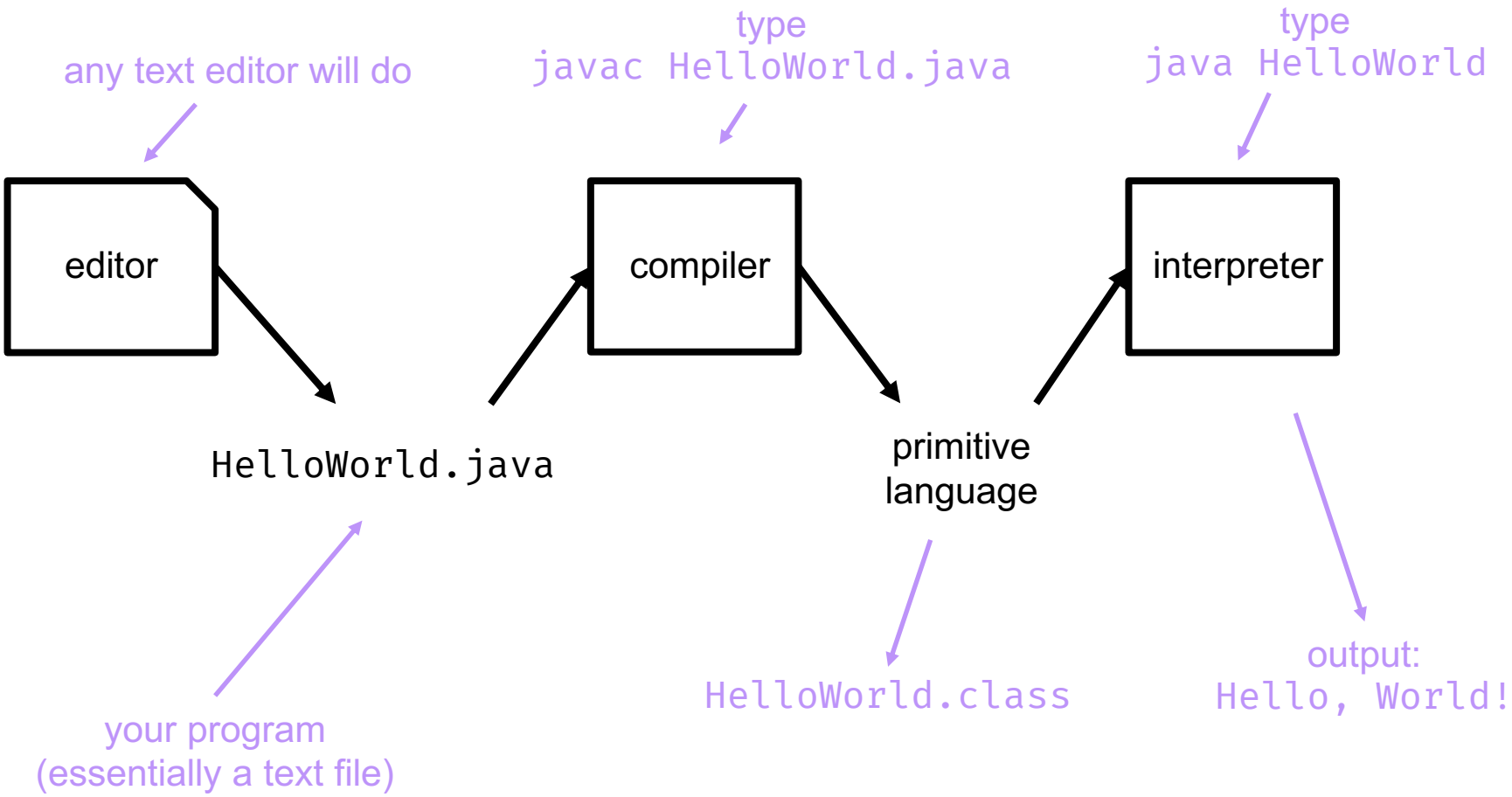
Running a Program

- Python is an interpreted language



Running a Program

- Java is an compiled language



Hello, World!

```
1  public class HelloWorld {
2    public static void main(String[] args) {
3      System.out.println("Hello, World!");
4    }
5  }
```

- Java is much more verbose and complicated…

Input & Output

Input / Output

- Program behavior typically dependent on changing circumstances

- Program code usually cannot be changed easily once you sold it

➢ Programs need to be able to react to *things*

➢ Modify **output** based on some **input**

- We already know how to do output: `print()`

Variables

- Variables are placeholders for values
- Computer stores information to be used later
- E.g.:

```
>>> x = 1
>>> print(x)
>   1
```

greeter.py

```
1  name = input("What is your name? ")
2
3  print("Hello, " + name)
```

- First Line:
  - use the `input` function to ask the user for their name and store the result in a variable called `name`
- Third Line
  - prints what we would expect…

Variables and Data Types

- Each variable has a specific type
  - python does not require us to specify the type at the moment
- Possible types are
  - Strings, ie. any sequence of characters
  - Numbers
    - Integer numbers, ie. 1, 2, 23, 42, 1337, -5, 0
    - Floating point numbers, ie. 1.5, -0.25, 3.14159265359, 2.7182818284
  - Booleans, ie. True and False
- Type specification via so-called *hints*:

```
>>> x: int = 1
>>> y: float = 3.14
```

- Data Types can be converted
  - called: *casting*
  - Two types of casting: implicit and explicit

- Explicit casting

```
>>> x: int = 1
>>> y: float = float(x)
>>> print(y)
    1.0
```

- Forces any value into the requested data type
  - to String: str(x)
  - to Integer: int(x)
  - to Float: float(x)
  - to Boolean: bool(x)

- Data Types can be converted
  - called: *casting*
  - Two types of casting: implicit and explicit

- Implicit casting
  - Silenty converts a value of one type to another type

```
>>> x: int = 1
>>> y: int = x + 1.0
>>> print(y)
```

When poll is active, respond at **pollev.com/christianbraune537**

What is the output of „print(y)"?

- TypeError
- 2
- 2.0
- "2.0"
- None of the above

Start the presentation to see live content. For screen share software, share the entire screen. Get help at **pollev.com/app**

- Data Types can be converted
  - called: *casting*
  - Two types of casting: implicit and explicit

- Implicit casting
  - Silenty converts a value of one type to another type

```
>>> x: int = 1
>>> y: int = x + 1.0
>>> print(y)
```

- Python silently changes the data type of `y` to `float`!

Arithmetics

- Variables alone are boring
  - A computer is called a computer because it *computes* things
- Combine variables to create new values
- Arithmetics (for numeric values):

```
• +      Addition            e.g. 3 + 2     = 5
• -      Subtraction         e.g. 0.5 + 0.5 = 1.0
• *      Multiplication      e.g. 2 * 3     = 6
• /      Division            e.g. 11 / 2    = 5.5
• //     Integer Division    e.g. 11 // 2   = 5
• %      Modulus/Remainder   e.g. 11 % 2    = 1
• **     Potentiation        e.g. 3 ** 2    = 9
```


Dr.-Ing. Christian Braune

Arithmetics

- Variables alone are boring
  - A computer is called a computer because it *computes* things
- Combine variables to create new values
- Arithmetics (for Strings):
  - `+  Concatenation  e.g. "foo" + "bar" = "foobar"`
  - `%  Formatting`
    `e.g. "Hello %s" % "Name" = "Hello Name"`
  - `*  Multiplication e.g. "a" * 3  = "aaa"`

Arithmetics

- Variables alone are boring
  - A computer is called a computer because it *computes* things
- Combine variables to create new values
- Arithmetics (for boolean values):
  - `&&` Conjunction (And) (also: and) e.g. `True && False = False`, `False && False = False`, `True && True = True`
  - `||` Disjunction (Or) (also: or) e.g. `True || False = True`, `False || False = False`, `True || True = True`
  - `^` Xor e.g. `True || False = True`, `False || False = False`, `True || True = False`
  - `!` Negation (Not) (also: not) e.g. `!True = False`, `!False = True`

Comparisons

- Variables alone are boring
  - A computer is called a computer because it *computes* things
- Combine variables to create new values
- Comparisons:
  - `<` strictly less than
  - `<=` less than (or equal)
  - `==` equal
  - `!=` not equal
  - `>=` greater than (or equal)
  - `>` greater than

Recap

- At this point we know how to do
  - output text

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

  - store information in variables

```
>>> x: int = 5
```

  - get input either by asking the user

```
>>> name = input("What is your name? ")
>>> print("Hello, " + name)
```

  - calculate new values from existing ones

```
>>> y: int = x + 5
```

Recap

- We are missing: Control Flow
- Change the behavior of our program based on some conditions
- Two main control flow techniques
  - Conditionals
  - Loops

Control Flow

Conditions & Loops

General Structure for a conditional

```
1  condition: boolean = True
2
3  if condition:
4      print("True!")
5  else:
6      print("False")
```

- Third Line: Conditional
  - The following block (everything that is indented by a multiple of 4 spaces) is executed if and only if the condition is true, here: the user typed 'password'
- Fifth Line:
  - The following block is executed if and only if the condition is evaluated to false

conditional.py

```
1  answer = input("Please enter 'password': ")
2
3  if answer != "password":
4      print("Wrong answer!")
5  else:
6      print("Correct! You may continue")
```

- Third Line: Conditional
  - The following block (everything that is indented by a multiple of 4 spaces) is executed if and only if the condition is true, here: the user typed 'password'
- Fifth Line:
  - The following block is executed if and only if the condition is evaluated to false

conditional2.py

```
1  answer = input("Please enter 'password': ")
2  answer2 = input("Speak, friend, and enter. ")
3
4  if answer != "password" && answer2 != "Mellon":
5      print("Wrong answer!")
6  else:
7      print("Correct! You may continue")
```

- Conditionals can be more complex
  - Arbitrary boolean expressions

conditional3.py

```
1  answer = input("Please enter 'password': ")
2
3  if answer == "password":
4      print("Correct!")
5  elif answer == "Password":
6      print("Correct! But please use lower case...")
7  else:
8      print("Wrong!")
```

- Fifth Line:
  - an extra conditional that is only executed if the first one fails
  - The else (line 7) is only executed if all `if/elif` before failed.

Loops

- Humans are smart and slow
- Computers are dumb and fast

➢ Repetitive tasks are great for computers

only_evens.py

```
x: int = 0
if x % 2 == 0:
    print(x)
x = x + 1
```

- Prints all even numbers from 0 to 100

only_evens2.py

```
1  x: int = 0
2  while x <= 100:
3      if x % 2 == 0:
4          print(x)
5      x = x + 1
```

- Does the same… But with much less code

- Third line:
  - special conditional
  - the following block is executed <u>as long as</u> the condition is `True`

For-Loops

```
x: int = 0
while some_variable_is_within_a_certain_range:
    # do_something
    increment_that_variable
```

- This pattern is extremely common
- There is a shorter notation for that

only_evens3.py

```
for x in range(101):
    if x % 2 == 0:
        print(x)
```

- Why 101?

$$\forall\, x\colon 0 \leq x < 101$$

- Usually lower bound is included and upper bound is excluded

only_evens3.py

```
1  for x in range(101):
2      if x % 2 == 0:
3          print(x)
```

- `range(upper)`

- Usually lower bound is included and upper bound is excluded

only_evens4.py

```
1  for x in range(10, 101):
2      if x % 2 == 0:
3          print(x)
```

- `range(lower, upper)`

$$\forall\, x: 10 \leq x < 101$$

- Usually lower bound is included and upper bound is excluded

only_evens5.py

```
for x in range(10, 101, 2):
    print(x)
```

- `range(lower, upper, step)`

- Usually lower bound is included and upper bound is excluded
- `step` defines the step size

Recap

Recap: What does this code snippet do?

```
1 print("Hello World!")
```

Recap: What does this code snippet do?

```
1  x = 5
2  y = 18.0
3
4  print(x + y)
```

Recap: What does this code snippet do?

```
1  x = input("Enter a number between 0 and 10: ")
2  x = int(x)  # Turn x into a number
3  if x < 0 or x > 10:
4      print("Wrong input")
5  print("Correct")
```

Recap: What does this code snippet do?

```
1  x = input("Enter a number between 0 and 10: ")
2  x = int(x)  # Turn x into a number
3  if x < 0 or x > 10:
4      print("Wrong input")
5  else:
6      print("Correct")
```

Recap: What does this code snippet do?

```
1  x = 1
2
3  while x < 0 or x > 10:
4      x = input("Enter a number between 0 and 10: ")
5      x = int(x)  # Turn x into a number
6  print(x)
```

Recap: What does this code snippet do?

```
1  x = -1
2
3  while x < 0 or x > 10:
4      x = input("Enter a number between 0 and 10: ")
5
6  print(x)
```

Time for a Game

A Game: FizzBuzz

- Count from 1 to 100
  - Say the number,
  - except if the number is divisible by 3,
    then say "FIZZ",
  - except if the number is divisible by 5,
    then say "BUZZ",
  - except if the number is both divisible by 3 and 5,
    then say "FIZZBUZZ"

A first program: FizzBuzz

- Count every number from 1 to 100
  - If the current number is divisible by 3 and 5 print "FIZZBUZZ"
  - If the current number is divisible by 3 print "FIZZ"
  - If the current number is divisible by 5 print "BUZZ"
  - If none of the above conditions are met, print the number

Time to try that on your own!

Cheat Sheet

- Count every number from 1 to 100
  - If the current number is divisible by 3 and 5 print "FIZZBUZZ"
  - If the current number is divisible by 3 print "FIZZ"
  - If the current number is divisible by 5 print "BUZZ"
  - If none of the above conditions are met, print the number

- Do something for all numbers in [lower, upper)

```
for number in
range(lower, upper):
```

- Check conditions:

```
if condition:
    ...
elif other_condition:
    ...
else:
    ...
```

- Arithmetics:
  - `+, -, *, /, %` (Addition, Subtraction, Multiplication, Division, Remainder)
  - &&, || (logical AND, logical OR)
- Output:
  - `print(...)`


TWO
HOURS LATER

A first program: fizzbuzz.py

```
for x in range(1, 101):
    if x % 3 == 0 and x % 5 == 0:
        print("FIZZBUZZ")
    elif x % 3 == 0:
        print("FIZZ")
    elif x % 5 == 0:
        print("BUZZ")
    else:
        print(x)
```

- Count every number from 1 to 100
  - If the current number is divisible by 3 and 5 print "FIZZBUZZ"
  - If the current number is divisible by 3 print "FIZZ"
  - If the current number is divisible by 5 print "BUZZ"
  - If none of the above conditions are met, print the number