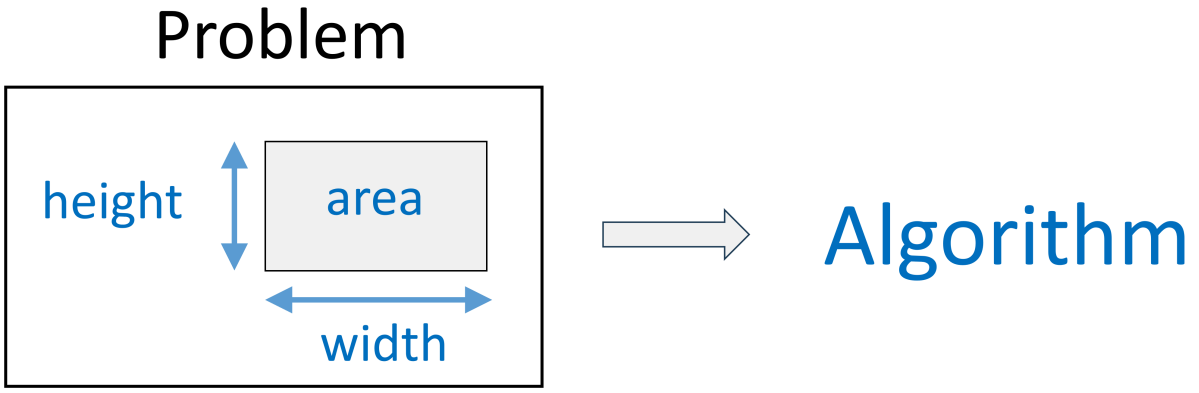
PHOT 110: Introduction to programming

LECTURE 02

Michaël Barbier, Spring semester (2023-2024)

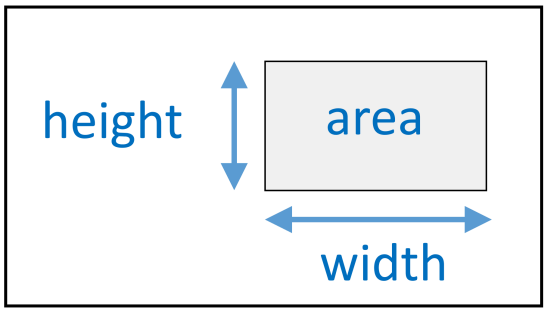
PYTHON BASICS & SYNTAX

TOWARDS PYTHON IMPLEMENTATIONS



TOWARDS PYTHON IMPLEMENTATIONS

Problem



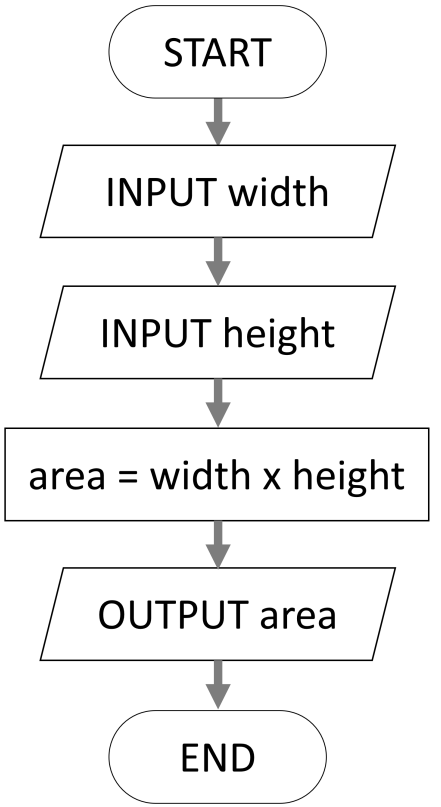
Algorithm

Pseudo code

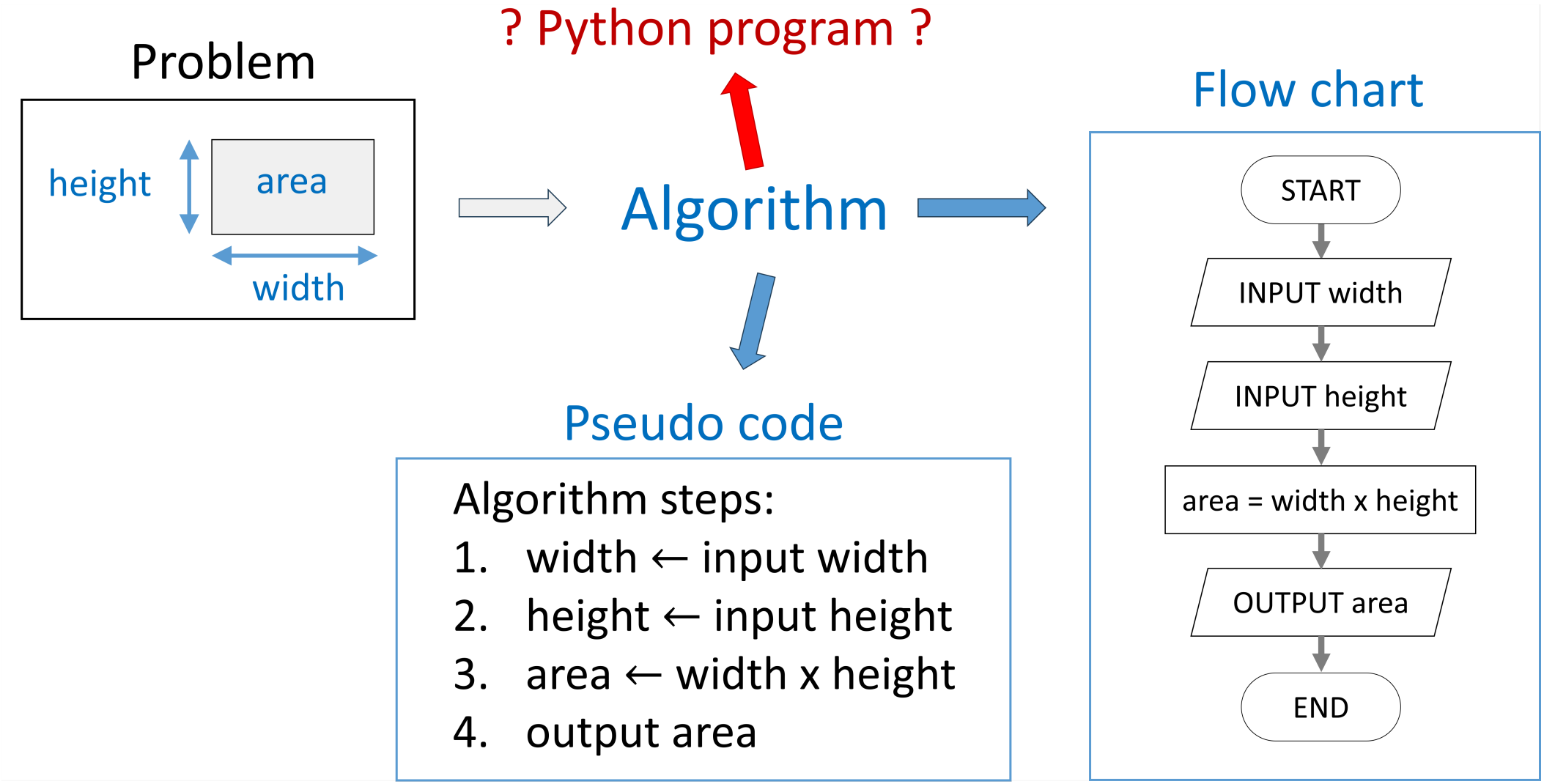
Algorithm steps:

1. width ← input width
2. height ← input height
3. area ← width x height
4. output area

Flow chart



TOWARDS PYTHON IMPLEMENTATIONS



PYTHON PROGRAM STRUCTURE

A Python script is a sequence of **statements** (algorithm steps), and **definitions** (functions, classes, …)

- Definitions are evaluated
- Statements are executed

Executing a program:

- Execute statements **one by one** in the **Python Console**
- Statements **stored** as a sequence in a **script file**

THE PYTHON CONSOLE

Execute statements one by one in the Python Console

```
>>> print(6)
6
>>> a = 10
>>>
```

Useful for:

- quick calculations
- testing the working of commands

PYTHON SCRIPTS

Statements stored in a script file

```
1 a = 10
2 b = 25
3 prod = a * b
4 print(f"{a} times {b} = {prod}")
```

```
10 times 25 = 250
```

Useful for:

- running a program multiple times
- incorrect lines can easily be found and corrected
- larger programs

DATA OBJECTS

- In Python everything is a data object: numbers, text, functions, class instances, etc.
- Statements perform actions on data objects

Data objects can be scalar:

Type	Description	(Example) values
`bool`	Boolean value	`True`, `False`
`int`	Integer numbers	.. , `-2`, `-1`, `0`, `-1`, `-2`, ..
`float`	Floating point	`3.56`, `23e-3`, `0.0079`
`NoneType`	Indicates no object	`None`

OBJECT TYPE

- Every object has a type

```
1 print(type(6.22e23))
```

```
<class 'float'>
```

- Possible to convert between certain types: type casting

```
1 it_is_raining = True
2 print(type(it_is_raining))
3 an_integer = int(it_is_raining)
4 print(type(an_integer))
```

```
<class 'bool'>
<class 'int'>
```

OBJECT TYPE

- automatic type casting in some cases

  `float <= int * float`

  `float <= int + float`

  `float <= int / int`

```
1  a = 4
2  b = 0.357
3  print( f"Type of (a * b) = {type(a * b)}" )
4  print( f"Type of (a / b) = {type(a / b)}" )
5  print( f"Type of (a + b) = {type(a + b)}" )
6  print( f"Type of (a / 25) = {type(a / 25)}" )
```

```
Type of (a * b) = <class 'float'>
Type of (a / b) = <class 'float'>
Type of (a + b) = <class 'float'>
Type of (a / 25) = <class 'float'>
```

VARIABLES

A **variable** is a name bound to an **object**:

- the object is **assigned** to the variable
- In pseudo code indicated by ←

In python assignment operator is the "=" sign:

```
1  a_variable_name = 4.5
```

The variable can be re-assigned another value or object:

```
1  v = 4
2  v = v + 2                          # v becomes 6
3  v = "Now v is assigned text"       # v has type str
```

EXPRESSIONS & OPERATORS

Objects can be combined by **operators** in **expressions**

Most used arithmetic operators:

symbol	description	example
**	Power	3**2 = 9
/	Division	5 / 4 = 1.25
*	Multiplication	3 * 4 = 12
+	Addition	5 + 7 = 12
-	Subtraction	6 - 9 = -3
%	modulo	34 % 6 = 4

EXPRESSIONS & OPERATORS

Most used logic operators:

symbol	description	example
`<`, `>`	smaller/larger than	`5 > 4` → `True`
`==`	is equal to	`3 == 6` → `False`
`<=`, `>=`	smaller/larger or equal	`5 <= 5` → `True`
`and`	boolean AND	`True and False` → `False`
`or`	boolean OR	`True or False` → `True`
`not`	boolean NOT	`not True` → `False`

EXPRESSIONS

An expression can be built with following rules:

- `<expression>` $\stackrel{def}{=}$ `<object>` (an object is an expression)
- `<expression>` $\stackrel{def}{=}$ `<operator> <expression>`
- `<expression>` $\stackrel{def}{=}$ `<expression> <operator> <expression>`

Expressions result into values and can be assigned to variables:

```
1 x = 4
2 y = 2*x**2 + 3*x - 5
3 print(f"The value of y = {y}")
```

```
The value of y = 39
```

OPERATOR PRECEDENCE AND BRACKETS

Precedence of operators is similar to mathematics.

List of precedence of operators can be found on the python.org website: https://docs.python.org/3/reference/expressions.html#operatc precedence

Round brackets can be used to give priority

```
1  x = 2
2  y = (12 - x) / 10
3  print(f"The value of y = {y}")
```

```
The value of y = 1.0
```

TEXT OBJECTS

Text objects are called strings: the type is `str` A string is defined by using single or double quotes:

```
1 "This is a string, single 'quotes' can be used here"
2 'Here we can use double "quotes" inside it'
```

Strings can also cover multiple lines, for that we use triple quotes:

```
1 """ This string runs over multiple lines.
2 Another line starts here.
3 And another one.
4 """
```

Also triple single quotes work as well.

OPERATORS WORKING ON TEXT OBJECTS

Appending one string to another with the “+” operator:

```
1 a_verb = "flies"
2 print("The balloon" + a_verb)
3 print("The balloon" + " " + a_verb)
```

```
The balloonflies
The balloon flies
```

Multiplying strings:

```
1 print(5 * "balloon ")
```

```
balloon balloon balloon balloon balloon
```

Casting a string to a number and vice versa:

```
1 print(5 * int("100"))
2 print(10 * str(50))
```

FORMATTING STRINGS AND NUMBERS

A formatted string is a string with prefix f or F:

```
1 the_radius = 25
2 formatted_str = f"A circle with radius {the_radius} m"
3 print(formatted_str)
```

```
A circle with radius 25 m
```

More complex formatting:

```
1 a = 1/6; b = 0.0145; c = 23e-6
2 print(f"The product {a} x {b} = {a * b}")
3 # Use format {variable:No_space.No_sign_digits}
4 print(f"{a:8.2} x {b:8.2} = {a * b:8.2}")
5 print(f"{a:8.2} x {c:8.2} = {a * c:8.2}")
```

```
The product 0.1666666666666666 x 0.0145 =
0.0024166666666666667
    0.17 x    0.015 =    0.0024
    0.17 x  2.3e-05 =   3.8e-06
```

PYTHON RESERVED WORDS OR KEYWORDS

- A list of words **reserved** for Python
- You cannot name variables (or functions/classes) similar

```
1 # List of keywords can be found in the keyword package
2 import keyword
3
4 print(keyword.kw_list)
```

```
['False', 'None', 'True', 'and', 'as', 'assert', 'async',
'await', 'break', 'class', 'continue', 'def', 'del', 'elif',
'else', 'except', 'finally', 'for', 'from', 'global', 'if',
'import', 'in', 'is', 'lambda', 'nonlocal', 'not', 'or',
'pass', 'raise', 'return', 'try', 'while', 'with', 'yield']
```

INPUT & OUTPUT (NOT IN INTERACTIVE MODE)

- `input()` can be used to ask user text input
- `print()` can be used to output text

```
# parameters
pizza_price = 200
extra_cheese_price = 20

# input accepts a string parameter
n_pizza = input("How many pizza's do you want: ")
extra_cheese = input("""Do you want extra cheese?\n
    For yes press [1]\n
    For no press [0]\n
Enter your choice""")

# Output to the user
print("Total cost: {int(n_pizza * (pizza_price + extra_cheese)}"
```

INDENTATION

- Is the amount of space preceding statements
- Indentation should be the same within a code block
- **Spaces** and **Tabs** are different
- Following code will give an `IndentationError: unexpected indent`

```
1  a = 4
2  b = 2
3   c = 3
4  y = a * (b + c)
```

```
IndentationError: unexpected indent (3759259151.py, line 3)
```

IMPORT STATEMENTS

Python packages can be imported in multiple ways

```
1  # import the package with its name
2  import math
3
4  area = 3**2 * math.pi
5  print(f"Area of a circle with radius 3 = {area}")
6
7  # import functions of the package separately
8  from math import sin, cos, pi, sqrt
9
10 angle = 23/180*pi
11 formula = sqrt(2/3) * sin(angle) + cos(angle)
12
13 # import packages or functions and rename them
14 from math import sqrt as sr
```

TIDY AND DOCUMENTED CODE

TIDY WRITING STYLE

- Writing **readable** code
- Clear and not too long variable names
- Use spaces in a consistent manner
- Use a Python code formatter such as **Black**: https://github.com/psf/black
- Document code both with **comments** and **docstrings**

COMMENT LINES

- Text of a line after a hash symbol is ignored
- The hash symbol can be in the beginning of the line:

```
1  # A comment on a single line
```

Comments can explain the next line of code

```
1  # Convert Matlab-indices to zero-based
2  ind = m_index - 1
```

Or the comment can start after a statement

```
1  y = 12 + x  # An inline comment: #-symbol after 2 spaces
```

MULTI-LINE COMMENTS

Multi-line strings can be used as comments, but

- they are not meant as comments (not ignored)
- they can become docstrings (documentation-strings)

```
1
2  '''
3  This multi-line string is not assigned to a
4  variable, therefore doesn't influence your code directly
5  '''
6
7  """
8  Docstrings consist of multi-line strings
9  but always use triple double quotes.
10 """
```

DOCSTRINGS

- Docstrings are multi-line strings providing documentation
- They populate the __doc__ variable
- The help() function uses the __doc__ variable

```
1 import math
2 print(math.__doc__)
```

```
This module provides access to the mathematical functions
defined by the C standard.
```

DOCSTRINGS

- Docstrings are multi-line strings providing documentation
- They populate the __doc__ variable
- The help() function uses the __doc__ variable

```
1 import math
2 help(math.sin)
```

```
Help on built-in function sin in module math:

sin(x, /)
    Return the sine of x (measured in radians).
```

- We will revisit Docstrings for functions and classes later

PROGRAM ERRORS

DIFFERENT ERROR-TYPES

Different types of errors can be encountered

- **Syntax errors**: code inconsistent with the Python language, the code will not run (i.e. not start).
- **Runtime errors**: exceptions occurring while the program runs, the code will crash.
- **Semantic errors**: the code does not what was intended

SYNTAX ERROR EXAMPLES

- Usage of unknown identifiers (variable, function, class)

```
1 positive_number = uint(45)
```

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

- Misaligned indentation

```
1 a = 34
2  b = 5
3 print(f"The sum is {a+b}")
```

```
IndentationError: unexpected indent (3752930843.py, line 2)
```

- Mistaken symbol “:” instead of “;”

```
1 a = 3: b = 5
```

```
SyntaxError: invalid syntax (173182802.py, line 1)
```

RUNTIME ERROR EXAMPLES

- Division by zero

```
1 fraction = 1 / 0
```

```
ZeroDivisionError: division by zero
```

- Type mismatch

```
1 netto_price = 230.50
2 kdv_ratio = 0.21
3 print("Price: " + netto_price + " + " + kdv_ratio + " kdv
```

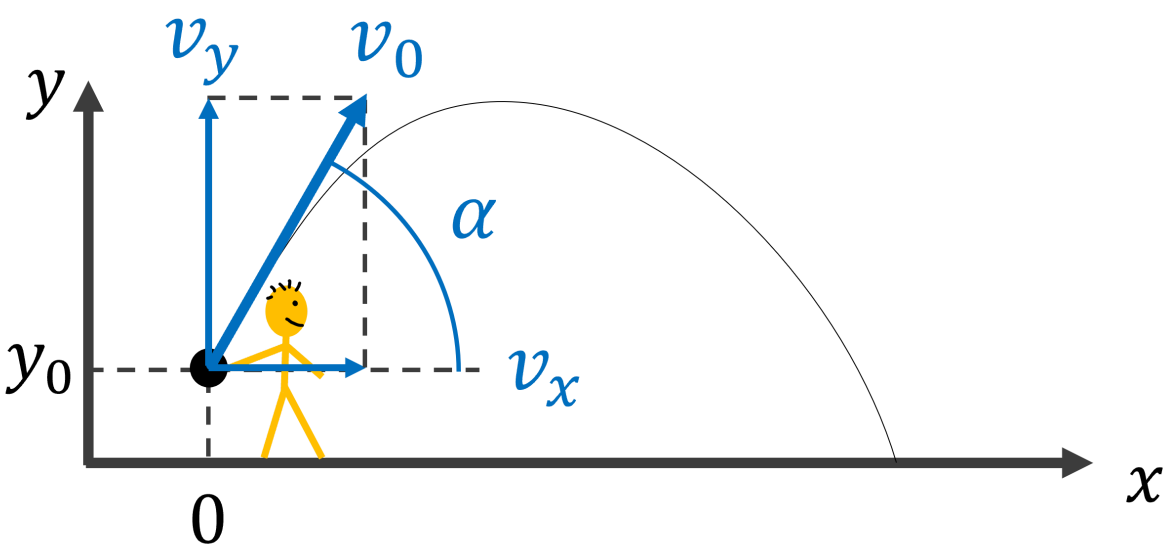
```
TypeError: can only concatenate str (not "float") to str
```

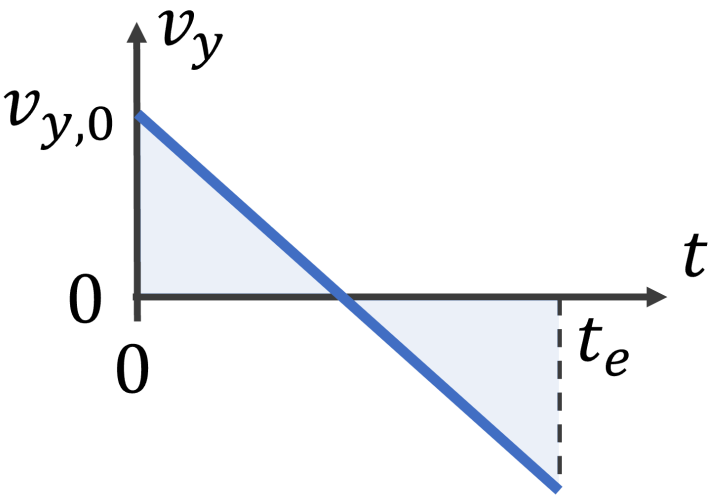
SUMMARY OF PYTHON BASICS

- Structure of a program in Python
  - Statements, expressions
- Python syntax
  - Known words
  - Proper indentation
- Commenting code
- Error types: Syntax, Runtime, and logical errors

SCIENTIFIC ALGORITHM: AN EXAMPLE

TRAJECTORY OF A BALL





- Gravitation: $g = 9.81$ m/s^2
- Air resistance: ignore for a slow heavy ball
- horizontal velocity is constant

HOW TO COMPUTE THE TRAJECTORY ?

The trajectory is a parabola (no air):

$$x = x_0 + v_{x,0}t$$

$$y = y_0 + v_{y,0}t - \frac{1}{2}gt^2$$

The ball will hit the ground at time t_e:

$$t_{e,\pm} = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a} = \frac{v_{y,0} \mp \sqrt{v_{y,0}^2 + 2gy_0}}{g}$$

TIME THAT THE BALL HITS THE GROUND

```
1  # Loading packages for sin, cos, pi, sqrt
2  import math
3
4  # Parameters of the trajectory
5  y0 = 1.20; x0 = 0
6  v0 = 8
7  alpha0 = 37 * (math.pi / 180)   # Angle
8  g = 9.81
9
10 # compute the time that the ball will hit the ground
11 vy0 = v0 * math.sin(alpha0)
12 vx0 = v0 * math.cos(alpha0)
13 te = (vy0 + math.sqrt(vy0**2 + 2*g*y0)) / g
14
```

```
The ball falls at time: 1.1875623706903884 s
```

HOW TO COMPUTE THE TRAJECTORY ?

The trajectory is a parabola (no air):

$$x = x_0 + v_{x,0} t$$

$$y = y_0 + v_{y,0} t - \frac{1}{2} g t^2$$

The equation for the parabola can be found by substitution. If $x_0 = 0$ then $t = x / v_{x,0} = x / (v_0 \cos(\alpha))$ and we obtain:

$$y = y_0 + \tan(\alpha)\, x - \frac{1}{2} \frac{g\, x^2}{v_0^2 \cos^2(\alpha)}$$

HOW FAR CAN WE THROW THE BALL ?

Let's start from the equation of the parabola:

$$y = y_0 + \tan(\alpha)\, x - \frac{1}{2}\frac{g\, x^2}{v_0^2 \cos^2(\alpha)}$$

When solving this quadratic equation for x we find:

$$x_{e,\pm} = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a} = \frac{\tan\alpha \mp \sqrt{\tan^2\alpha + 2gy_0/(v_0\cos\alpha)^2}}{g/(v_0\cos\alpha)^2}$$

Where the largest root is the throwing distance.

HOW FAR CAN WE THROW THE BALL ?

```
1  # Loading packages for sin, cos, pi, sqrt
2  from math import sin, cos, pi, sqrt, tan
3
4  # Parameters of the trajectory
5  y0 = 1.20; x0 = 0
6  v0 = 8
7  alpha0 = 37 * (pi / 180)  # Angle in radians
8  g = 9.81
9
10 # compute the time that the ball will hit the ground
11 vy0 = v0 * sin(alpha0)
12 vx0 = v0 * cos(alpha0)
13 xe_num = (tan(alpha0) + sqrt(tan(alpha0)**2 + 2*g*y0 / (v
14 xe_den = (g / (v0 * cos(alpha0))**2)
```

```
The ball falls at x: 7.587435837034325 m
```

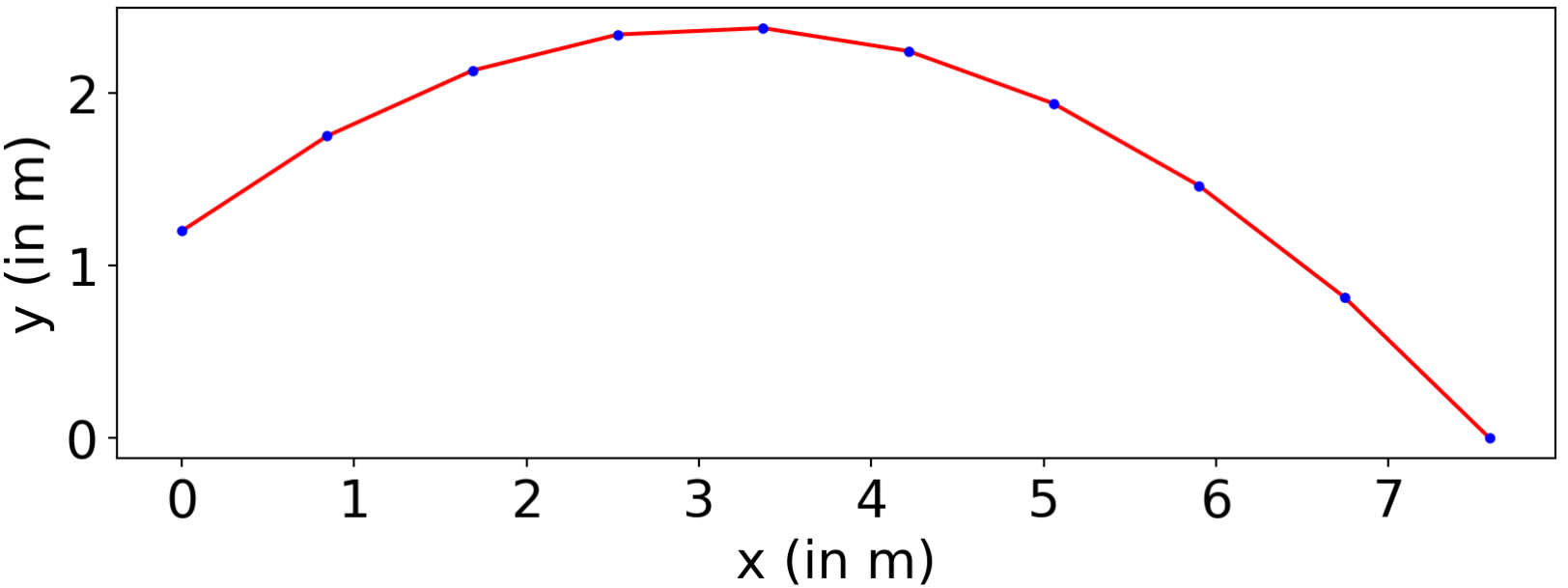
TRAJECTORY OF A BALL

```
# Loading packages for plotting and numeric calc.
import matplotlib
import matplotlib.pyplot as plt
import numpy as np

# Computing x and y coordinates for the trajectory
ts = np.linspace(0, te, 10)
xs = vx0 * ts
ys = y0 + (vy0 * ts) - (g*ts**2)/2

# Plotting the trajectory of the ball
matplotlib.rcParams.update({'font.size': 20})
plt.plot(xs, ys, "-", color="red")
plt.plot(xs, ys, ".", color="blue")
```

TRAJECTORY OF A BALL



Lecture 02: Python basics