Lecture 2, Part 2: Programming in Python

Course outline

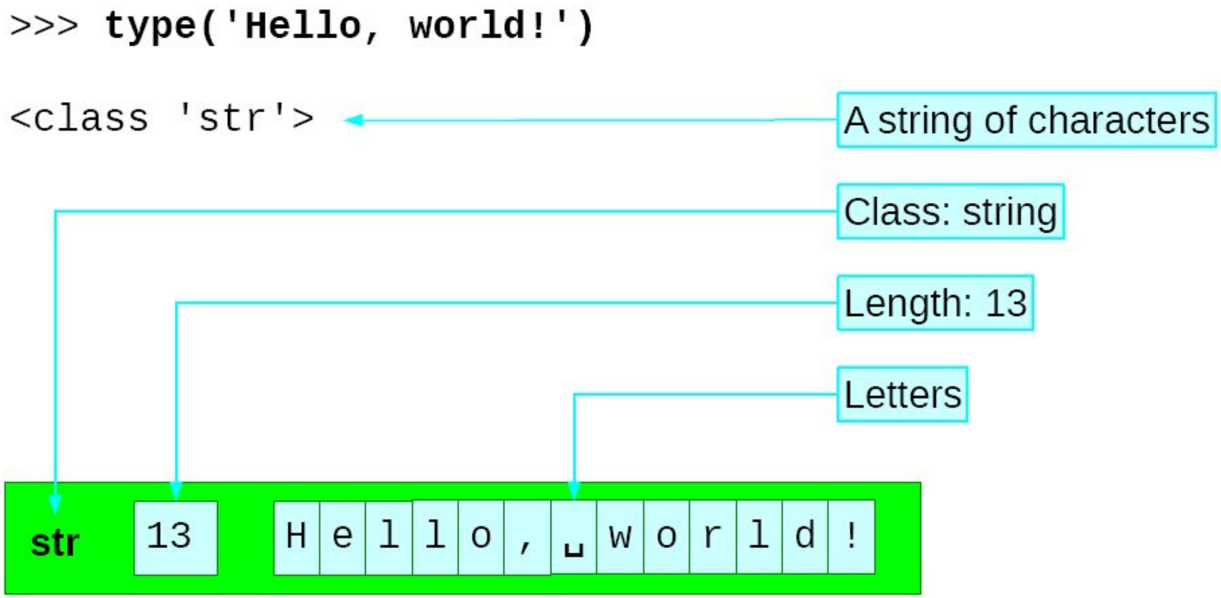
- Part 1 Introduction to Computing and Programming (first 2 weeks):
  - Problem solving: Problem statement, algorithm design, programming, testing, debugging
  - Scalar data types: integers, floating point, Boolean, others (letters, colours)
  - Arithmetic, relational, and logical operators, and expressions
  - Data representation of integers, floating point, Boolean
  - Composite data structures: string, tuple, list, dictionary, array
  - Sample operations on string, tuple, list, dictionary, array
  - Algorithms (written in pseudo code) vs. programs
  - Variables and constants (literals): association of names with data objects
  - A language to write pseudo code
  - Programming languages: compiled vs. interpreted programming languages
  - Python as a programming language
  - Computer organization: processor, volatile and non-volatile memory, I/O

Course outline (may change a bit)

- Part 2 Algorithm design and Programming in Python (balance 11 weeks):
  - Arithmetic/Logical/Boolean expressions and their evaluations in Python
  - Input/output statements (pseudo code, and in Python)
  - Assignment statement (pseudo code, and in Python)
  - Conditional statements, with sample applications
  - Iterative statements, with sample applications
  - Function sub-programs, arguments and scope of variables
  - Recursion
  - Modules
  - Specific data structures in Python (string, tuple, list, dictionary, array), with sample applications
  - Searching and sorting through arrays or lists
  - Handling exceptions
  - Classes, and object-oriented programming
  - (Time permitting) numerical methods: Newton Raphson, integration, vectors/matrices operations, continuous-time and discrete-event simulation

String and string operations

- A string is a sequence of characters
- Strings are enclosed in single quotes <u>or</u> double quotes
- Strings are of type ‘str’



```
>>> type('Hello, world!')

<class 'str'>
```

ASCII characters

- Character encoding is necessary to be able to represent them in binary form
- Two popular encoding schemes: ASCII and Unicode
- ASCII:
  - It can represent 128 characters:
    - 96 printable characters including English/Latin letters, punctuation marks
      - i.e. **a**, …, **z**, **A**, …, **Z**, **#**, **%**, **@**, etc.
    - 32 control characters (such as **SOH** ASCII 1), **STX** ASCII , **ETX** ASCII 3)
  - the 8-th bit is the parity check
- Unicode:
  - Supports more than 120,000 different characters
  - UTF-8, UTF-16, UTF-32 are some of the Unicode encoding schemes
  - UTF-8 and ASCII are fully aligned

- Python by default uses UTF-8

International characters using UTF-16

- Standard for encoding text expressed in most of the world's scripts
- Covering 154 modern and historic scripts
- 143,859 characters
- Uses ’\u’ followed by the hexadecimal (base 16) code for character
- Examples:

```
>>> print('\u011f‘)
'ğ‘
>>> '\u0915‘
'क‘
>>> '\u0950‘
'ॐ‘
>>> '\u0967‘
'१‘
```

Read
Unicode 16 for Devanagari script
About Unicode organization

ASCII and Unicode

String and string operations

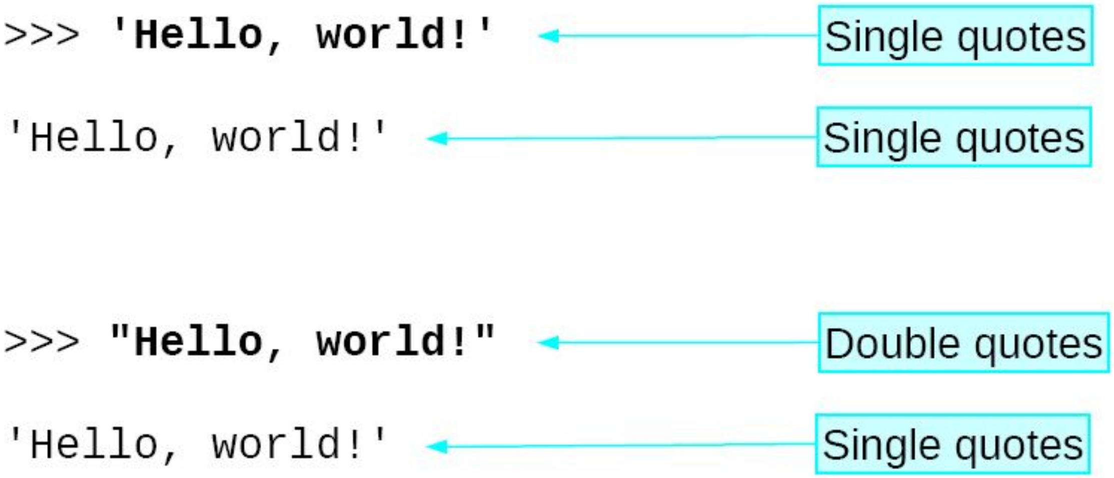
- A string is a sequence of characters
- Strings are of type ‘str’
- Strings are enclosed in single quotes <u>or</u> double quotes

```
>>> type(‘Hello, world!’)  <class ‘str’>
```

- Operators ‘**+**’ and ‘*****’
  - **`‘Hello’ + ‘ ‘ + ‘World!’  ‘Hello World!’`** (‘**+**’ is for concatenation)
  - **`‘John’*2`** ‘JohnJohn’ (* is for repetition)
  - Try out **`2*‘John’`**, and see what happens
    - Useful to draw a line **`10*’-’`** will give **`‘----------’`**

String and string operations

- Strings are enclosed in single quotes or double quotes



```
>>> 'Hello, world!'
'Hello, world!'


>>> "Hello, world!"
'Hello, world!'
```

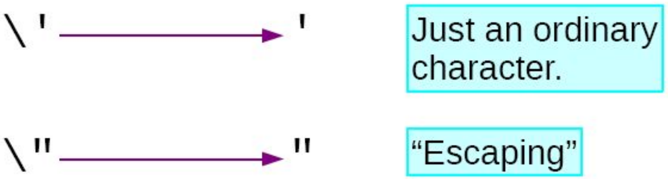
String and string operations

- String that contains single quotes <u>or</u> double quotes

```
>>> print('He said "hello" to her.')

He said "hello" to her.
```

```
>>> print("He said 'hello' to her.")

He said 'hello' to her.
```

```
>>> print('He said \'hello\' to her.')

He said 'hello' to her.
```



$\backslash' \rightarrow '$

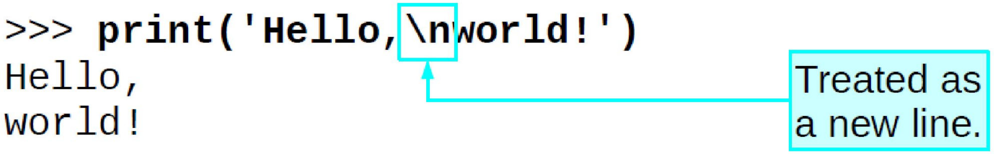
$\backslash" \rightarrow "$

Just an ordinary character.

“Escaping”

String and string operations

- Inserting special characters



```
>>> print('Hello,\nworld!')
Hello,
world!
```

Treated as a new line.

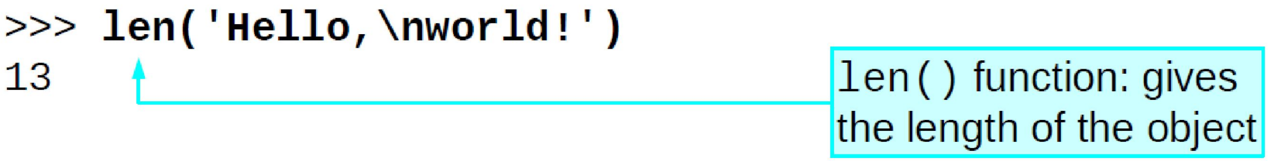
Strings and string operations

String and string operations

- Length of string:
  - `len(s)`
- Indexing
  - An “index” is used to refer to and access individual character
  - Example:
    - ‘John’[0]
    - ‘John’[3]
    - ‘John’[4] -- **IndexError: string index out of range**
    - ‘John’[-1]
    - ‘John’[:]
    - ‘John’[:2]

String and string operations

- Length of string:
  - **`len(s)`**
  - Example:
  - **`len(‘Hello’)`** is 5, indexed from 0 through 4



```
>>> len('Hello,\nworld!')
13
```

len() function: gives the length of the object

String, and string operations

- Slicing a string == extracting a substring
- General syntax is

`s[start:end:step]`

where

`start`: index to start slicing the string

`end`: string is sliced until end-1

`step`: determines the increment/decrement between each index for slicing

Examples:

```
>>> s1 = "Hello World“
>>> print(s1[4:11:2])
‘oWrd’

>>> s2 = "Hello“
>>> print(s2[1:len(s2):1]) # same as print(s2[1:5:1])
‘ello’

>>> s3 = "Hello Howdee?“
>>> print(s3[0:-1:1])
‘Hello Howdee’
>>> print(s3[-1])
‘?’
```

String, and string operations

- Length of string:
  - **`len(s)`**
- Indexing
  - An “index” is used to refer to and access individual or many characters in a string
  - Examples:

```
>>>‘John’[0]
J

>>>‘John’[3]
n

>>>’John’[4]
-- IndexError: string index out of range since len(‘John’) is 4

>>>‘John’[-1]
n

>>>‘John’[:]
John

>>>‘John’[:2]
Joh
```

Conditional statements

- Pseudo code:

```
if C1 then S1
```

- In Python:

```
if C1:
      ␣␣␣␣S1
```

- Example:

```
INC = float(input(‘Your Income? '))
Tax = 0
if INC > 100000:
    Tax = 0.1*(INC-100000)
print(‘Income is ‘, INC, ‘Tax is ‘, Tax)
```

Conditional statements

- Pseudo code:

```
if C1 then S1
```

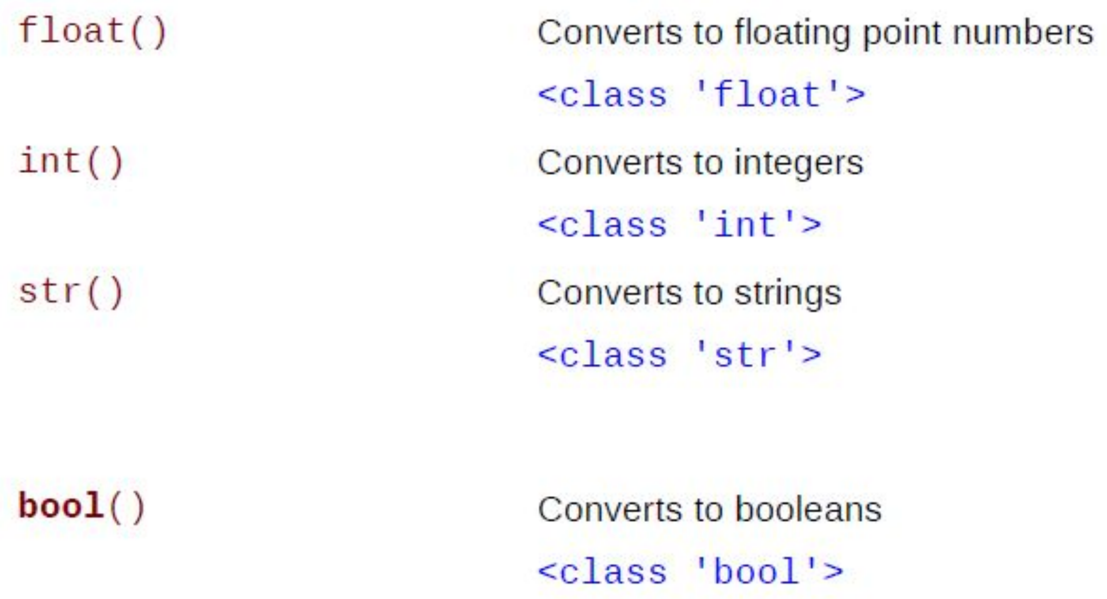
- In Python:

```
if C1:
    S1
```

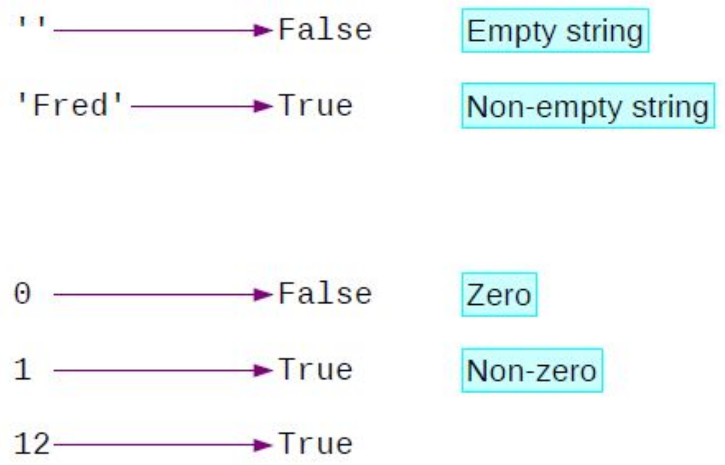
- Example:

```
INC = float(input(‘Your Income? '))
Tax = 0
if INC > 100000:
    Tax = 0.1*(INC-100000)
print(‘Income is ‘, INC, ‘Tax is ‘, Tax)
```

Conversion between data types



`float()`	Converts to floating point numbers `<class 'float'>`
`int()`	Converts to integers `<class 'int'>`
`str()`	Converts to strings `<class 'str'>`
`bool()`	Converts to booleans `<class 'bool'>`



'' → False Empty string

'Fred' → True Non-empty string

0 → False Zero

1 → True Non-zero

12 → True

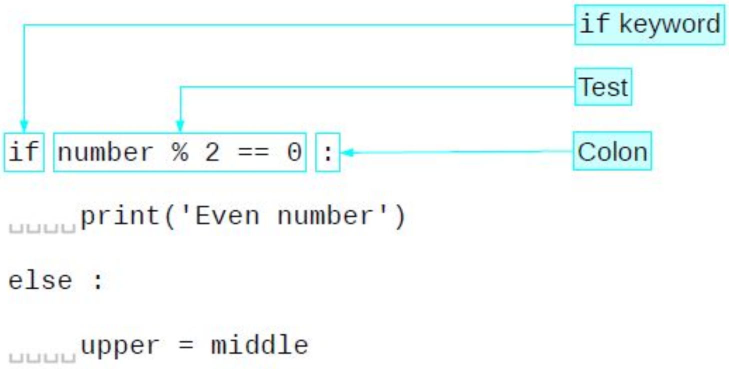
Conditional statements

- Pseudo code:

```
if C1 then S1 else S2
```

- In Python:

```
if C1:
  S1
else:
  S2
```



```
if number % 2 == 0 :
␣␣␣␣print('Even number')
else :
␣␣␣␣upper = middle
```

- Example:

```
T1 = float(input('Time 1? '))
T2 = float(input('Time 2? '))
print('T1, T2 ', T1, T2)
if(T1 < T2):
    minT = T1
else:
    minT = T2
print(T1, T2, minT)
```

Conditional statements

- Pseudo code:

```
if C1 then S1 else S2
```

- In Python:

```
if C1:
  S1
else:
  S2
```

- Example:

```
T1 = float(input('Time 1? '))
T2 = float(input('Time 2? '))
print('T1, T2 ', T1, T2)
if(T1 < T2):
    minT = T1
else:
    minT = T2
print(T1, T2, minT)
```

Input multiple data items

One way to input no. of data items:

```
>>>Input(‘x= ‘)
x= 123
>>>print(x)
123
>>>Input(‘y= ‘)
y= 345
>>>print(y)
345
```

Another way to input multiple data items:

```
>>> x, y = float(input(‘x?’)), float(input(‘ y? ‘))
x? 123 y? 345
>>> print('x = ', x, 'y = ', y)
x = 123.0 y = 345.0
>>>
```

Input multiple data items

Yet another way to input no. of data items

```
# taking multiple inputs at a time
>>>x, y, z = input("Enter a three value: ").split()
Enter a three value: 23 14 9

>>>print("Number of books in English: ", x)
Number of books in English: 23

>>>print("Number of books in Hindi: ", y)
Number of books in Hindi: 14

>>>print("Number of books in Urdu: ", z)
Number of books in Urdu: 9
>>>
```

split() method to split a Python string using a “separator” (e.g. “space”)

Conditional statements

- Pseudo code:

```
if C1 then S1 else [if C2 then S2]
```

- In Python:

```
if C1:
     S1
elif C2:
 S2
```

- Example:

```
INC = float(input(‘Your Income? '))
Tax = 0
if INC > 200000:
  Tax = 10000 + 0.2*(INC-200000)
elif INC > 100000:
  Tax = 0.1*(INC-100000)
print(‘Income is ‘, INC, ‘Tax is ‘, Tax)
```

Conditional statements

- Pseudo code:

```
if C1 then S1 else [if C2 then S2]
```

- In Python:

```
if C1:
  S1
elif C2:
  S2
```

- Example:

```
  if x%2 == 0:
      if x%3 == 0:
          print(x, ‘is divisible by 2 and 3’)
      else:
          print(x, ‘is divisible by 2 but not by 3’)
  elif x%3 == 0:
      print(x, ‘is divisible by 3 but not by 2’)
```

Iteration

Python supports while and for loops

- Pseudo code

```
while C then S
```

- In Python

```
while C:
␣␣␣␣
         S
```

Iteration

Python supports while and for loops

- Pseudo code

```
# Find the largest n such that 2**n ≤ 1000
n = 0; x = 2**n;
while x ≤ 1000 do [n = n+1; x = 2**n];
output(‘largest n such that 2**n ≤ 1000 is ‘, n-1)
```

- In Python

```
       # Find the largest n such that 2**n ≤ 1000
   n = 0
x = 2**n
   while x <= 1000:
       n = n+1
       x = 2**n
   print(‘largest n such that 2**n <= 1000 is‘, n-1)
```

- Question: what will be the output?

Iteration

Python supports while and for loops

- Execution of Python code:

```
   #
# Find the largest n such that 2**n ≤ 50
n = 0
x = 2**n
while (x <= 50):
    n = n+1
    x = 2**n
print("largest n such that 2**n <= 50 is", n-1)
```

- To determine what will be the output:

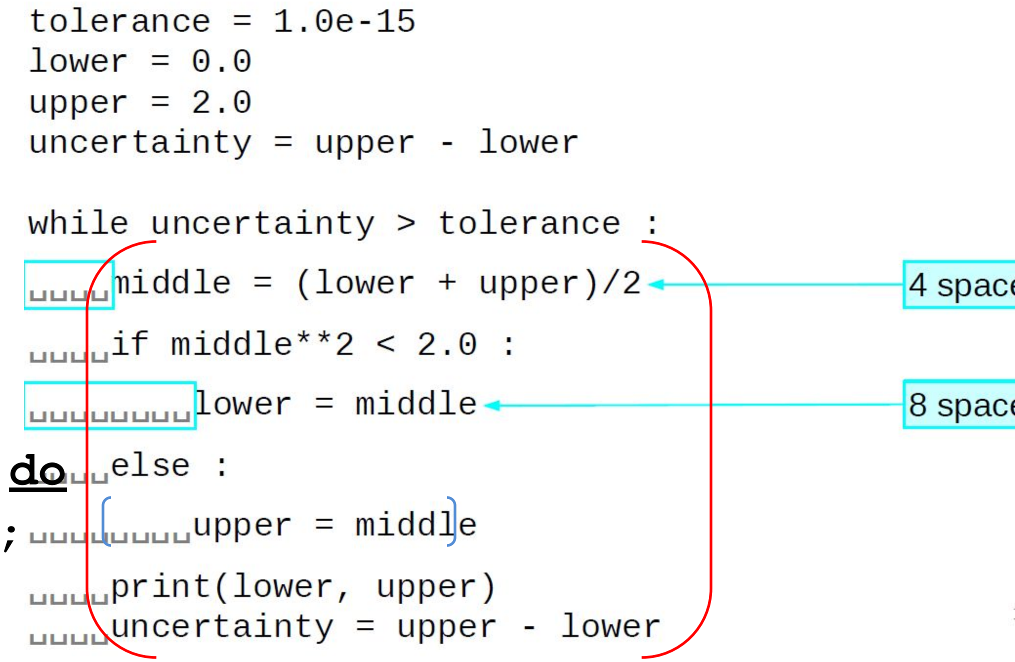
Test of condition	n	x	x<=50
1 st	0	1	TRUE
2 nd	1	2	TRUE
3 rd	2	4	TRUE
4 th	3	8	TRUE
5 th	4	16	TRUE
6 th	5	32	TRUE
7 th	6	64	FALSE

A note on indentation

Beware: indentation matters:

In pseudo code:

```
# compute the SQRT of 2.0
tolerance = 1.0 e-15;
lower = 0.0;
upper = 2.0;
uncertainty = upper-lower;
while uncertainty > tolerance do
    [middle = (lower + upper)/2;
    if middle**2 < 2.0
    then lower = middle
    else upper = middle;
    print(lower, upper);
    uncertainty = upper-lower
    ]
```



```
tolerance = 1.0e-15
lower = 0.0
upper = 2.0
uncertainty = upper - lower

while uncertainty > tolerance :
␣␣␣␣middle = (lower + upper)/2
␣␣␣␣if middle**2 < 2.0 :
␣␣␣␣␣␣␣␣lower = middle
␣␣␣␣else :
␣␣␣␣␣␣␣␣upper = middle
␣␣␣␣print(lower, upper)
␣␣␣␣uncertainty = upper - lower
```

Iteration – break command

- **break** commend
  - Used to terminate the loop when **break** statement is encountered
  - Improves efficiency (need not wait until loop terminates)
  - Control is transferred to statement following loop

- Example

```
#Find a positive integer divisible by both 11 and 12
x = 1
while True:
    if x%11 == 0 and x%12 == 0:
        break
    x = x + 1
print(x, "is divisible by 11 and 12")
```

Output:

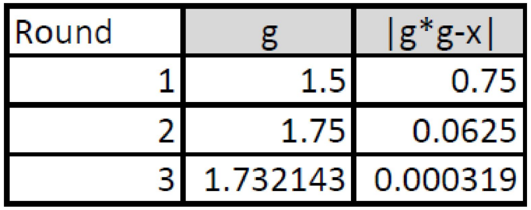
```
132 is divisible by 11 and 12
```

Iteration

Example: computing square root y = √x, where x > 0

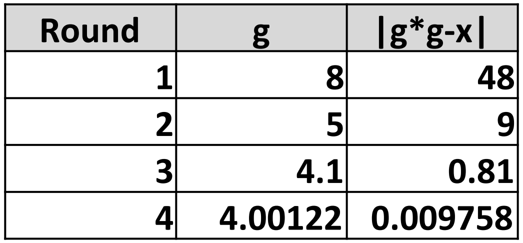
Somewhat informal version of an algorithm

```
    Start with a guess, g = x/2 # for instance
    if |g*g – x| is small
    then [conclude g = √x; output(g); stop]
    else [compute new guess g = (g + x/g)/2; repeat step 2]
Example outcome from above after 3 rounds:
Let x = 3
```



Round	g	|g*g-x|
1	1.5	0.75
2	1.75	0.0625
3	1.732143	0.000319

`or x = 16`



Round	g	|g*g-x|
1	8	48
2	5	9
3	4.1	0.81
4	4.00122	0.009758

Iteration

Example: computing square root g = √x, where x > 1

Another algorithm, based on “bisection method”

```
#compute sqrt(x), where x>1
x = input()
epsilon = 0.0001
low = 0
high = x
g = (low+high)/2 #initial guess
while abs(g*g - x) >= epsilon:
 if g*g < x:
     low = g
 else:
     high = g
 g = (low+high)/2 #new better guess
print(g)
```

Iteration

Example: computing square root x = √k, k > 0, or solving equation x^2 - k = 0

Another algorithm, based on “Newton-Raphson method”

```
#compute sqrt(k), where k > 0
k = input()
epsilon = 0.0001
x = k/2 #initial guess
while abs(x*x - k) >= epsilon:
 x = x - ((x*x - k)/(2*x)) #new better guess
print(x)
```

Q&A

- **?**