Algorithm Design Strategies I

Joaquim Madeira

Version 0.4 – September 2024

Overview

- Python 3 – A quick review
- Algorithm Efficiency – Counting basic operations

[python.org]

PYTHON 3

Python – Main features

- Dynamic typing

Interpreted Language

- Simple, consistent syntax and semantics
- Multiplatform
- Highly modular
- Suited for rapid development and large-scale programming

Python – Main features

- Reasonably fast and easily extended with C or C++ modules for higher speeds
- Easy access to various GUI toolkits
- Built-in advanced features, such as advanced hash tables, …
- Powerful included libraries such as numeric processing, image manipulation, …

Python – Pros

- Easy to learn / use
  - Familiar constructs
  - Very simple rules
  - Types associated with objects, not variables
- Much higher level of abstraction
  - Extensive standard core library
- Well-suited for rapid application development

Python – Pros

- Expressive
  - Comparatively fewer lines of code
  - Faster development
  - Easier maintenance and debugging

```
var2, var1 = var1, var2
```

Python – Pros

- Readable
  - Use of indentation !!

```
def pairwise_sum(list1, list2):
    result = []
    for i in range(len(list1)):
        result.append(list1[i] + list2[i])
    return result
```

Python - Pros

- Complete
  - “Batteries included” – Standard library
  - Everything you need to do real work
- Cross-platform

Python – Cons

- Not the fastest language…
  - Semicompiled to an internal byte-code form
  - Executed by a Python interpreter
  - BUT, computers have so much computing power
- For most applications, running speed isn’t as important as development speed

Python – Cons

- Not the most libraries
  - Java has a larger library collection
  - BUT, Python is easy to expand
- Not checking variable types at compile time
  - Variables are not bound to particular types
  - Not catching variable type mismatches

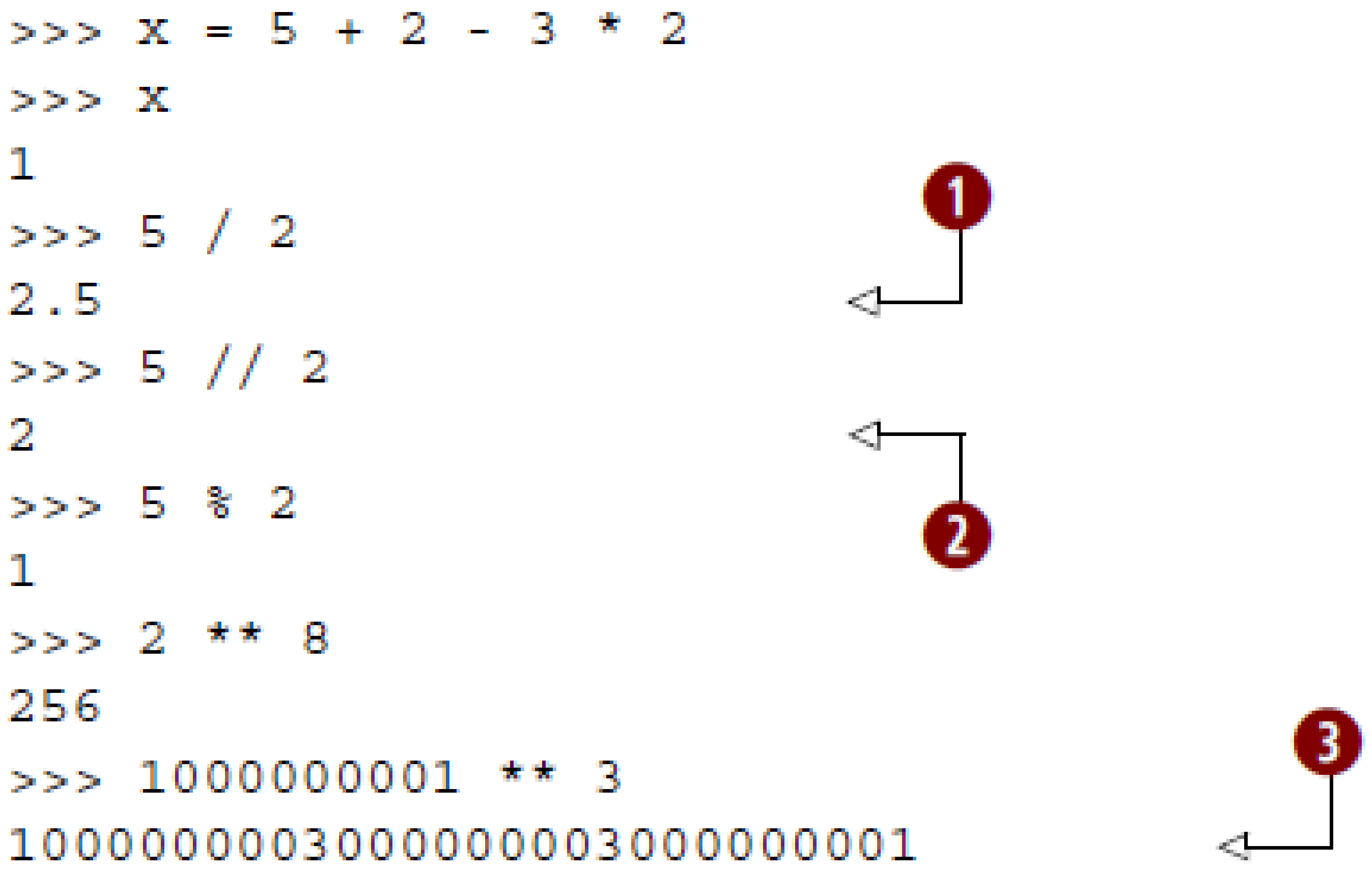
Python 3 – Why?

- More consistent, readable and less ambiguous than Python 2
- BUT, no full compatibility with older code
  - Fairly small changes
  - Strategies for code migration
- No longer a relevant issue…

Python 3 – Numbers

- Integers
  - 1, –3, 42, 355, 888888888888888, –7777777777
- Floats
  - 3.0, 31e12, –6e–4
- Complex numbers
  - 3+2j, –4– 2j, 4.2+6.3j
- Booleans
  - True, False

Python 3 – Integers

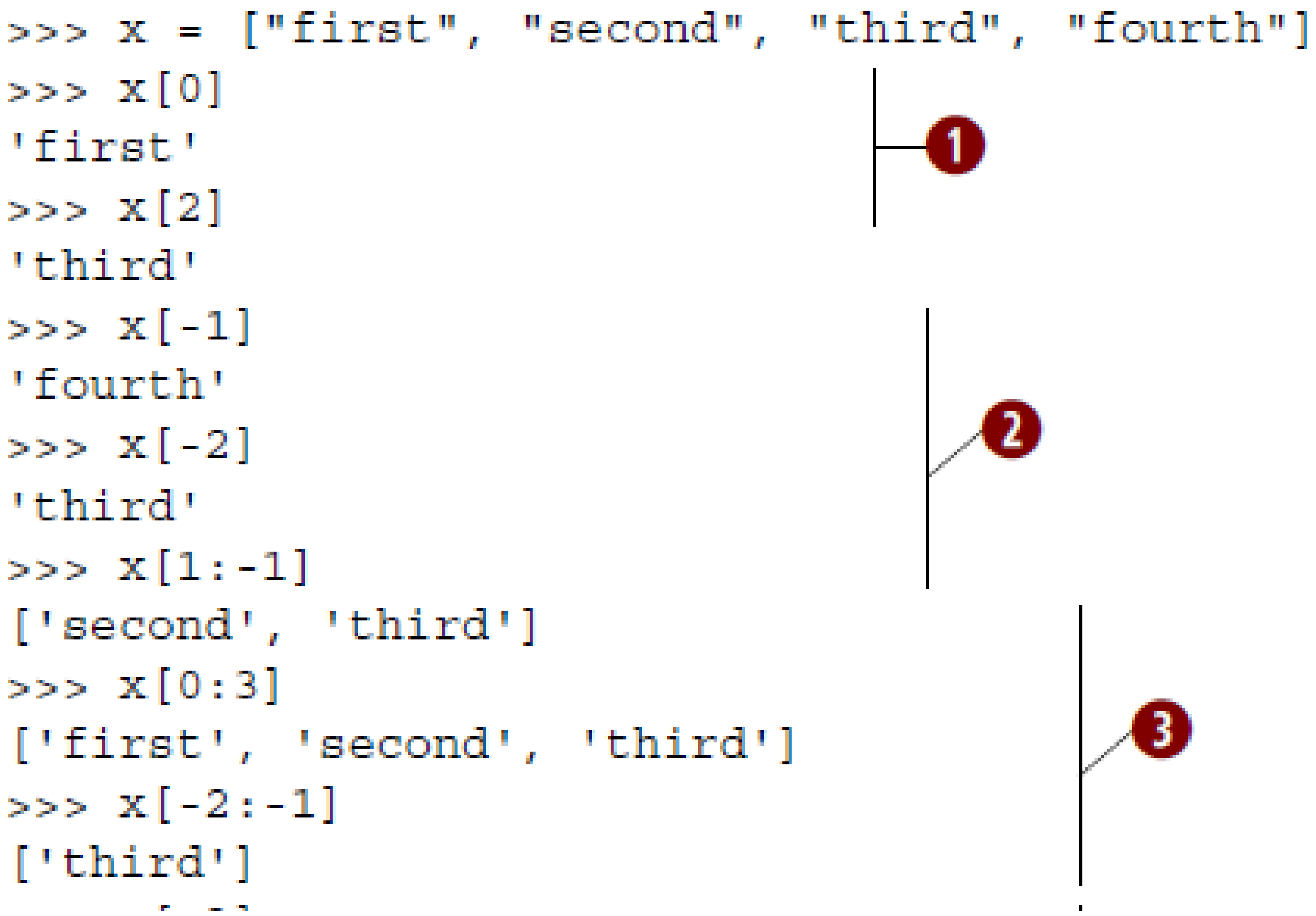


```
>>> x = 5 + 2 - 3 * 2
>>> x
1
>>> 5 / 2
2.5
>>> 5 // 2
2
>>> 5 % 2
1
>>> 2 ** 8
256
>>> 1000000001 ** 3
1000000003000000003000000001
```

Python 3 – Lists

```
[]
[1]
[1, 2, 3, 4, 5, 6, 7, 8, 12]
[1, "two", 3L, 4.0, ["a", "b"], (5,6)]
```

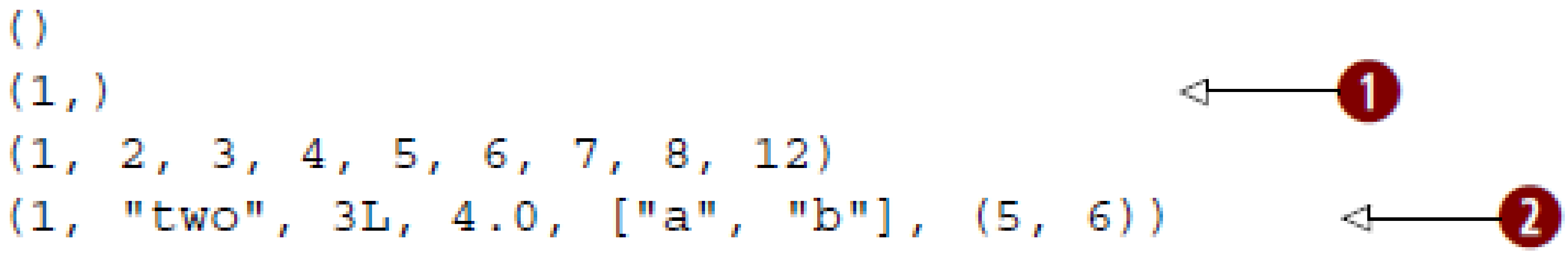
Indexing and slicing



```
>>> x = ["first", "second", "third", "fourth"]
>>> x[0]
'first'
>>> x[2]
'third'
>>> x[-1]
'fourth'
>>> x[-2]
'third'
>>> x[1:-1]
['second', 'third']
>>> x[0:3]
['first', 'second', 'third']
>>> x[-2:-1]
['third']
```

Python 3 – Tuples

- Tuples are immutable !!



```
()
(1,)                                          ← 1
(1, 2, 3, 4, 5, 6, 7, 8, 12)
(1, "two", 3L, 4.0, ["a", "b"], (5, 6))       ← 2
```

```
>>> x = [1, 2, 3, 4]
>>> tuple(x)
(1, 2, 3, 4)
```

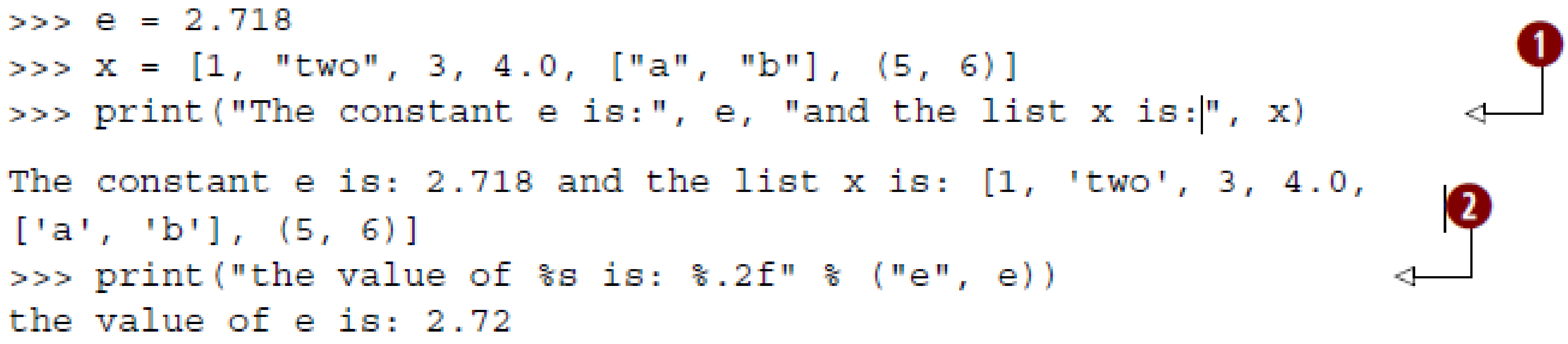
```
>>> x = (1, 2, 3, 4)
>>> list(x)
[1, 2, 3, 4]
```

Puthon 3 - Strings

- Strings are also immutable !!

```
"A string in double quotes can contain 'single quote' characters."
'A string in single quotes can contain "double quote" characters.'
'''\This string starts with a tab and ends with a newline character.\n'''
"""This is a triple double quoted string, the only kind that can
    contain real newlines."""
```

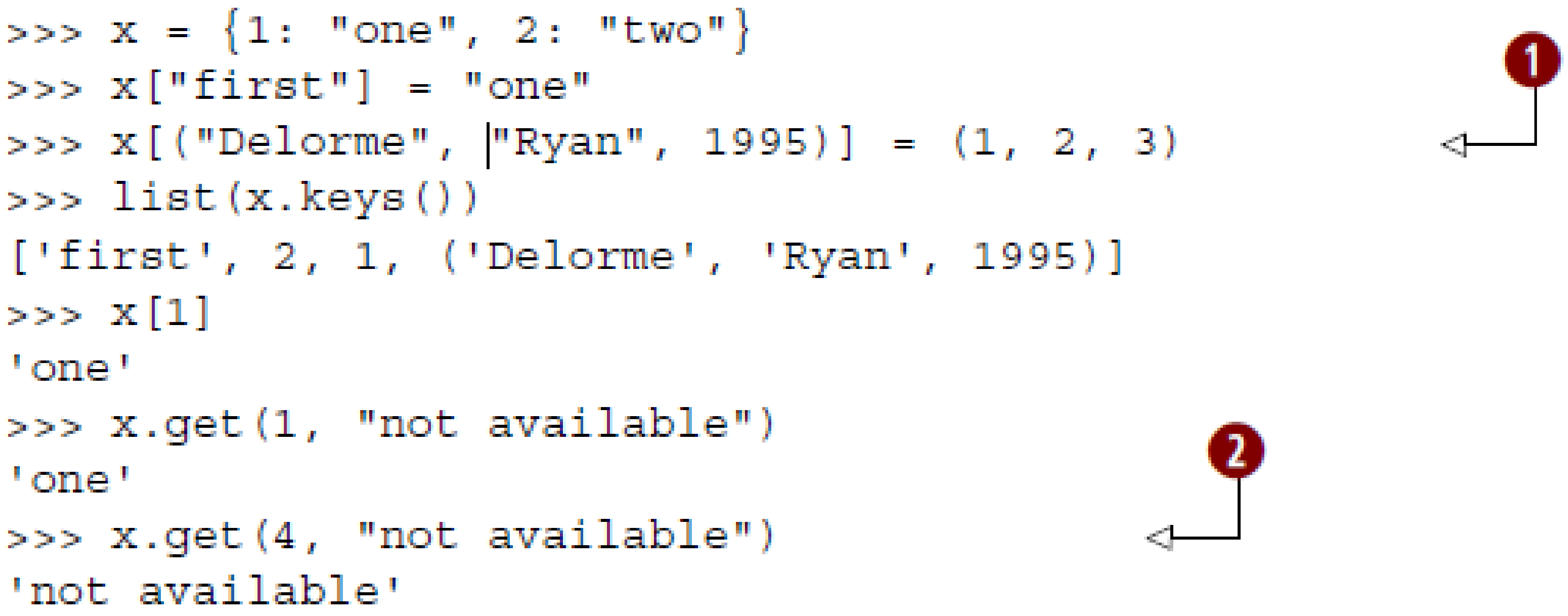
- The print function outputs strings



```
>>> e = 2.718
>>> x = [1, "two", 3, 4.0, ["a", "b"], (5, 6)]
>>> print("The constant e is:", e, "and the list x is:", x)

The constant e is: 2.718 and the list x is: [1, 'two', 3, 4.0,
['a', 'b'], (5, 6)]
>>> print("the value of %s is: %.2f" % ("e", e))
the value of e is: 2.72
```

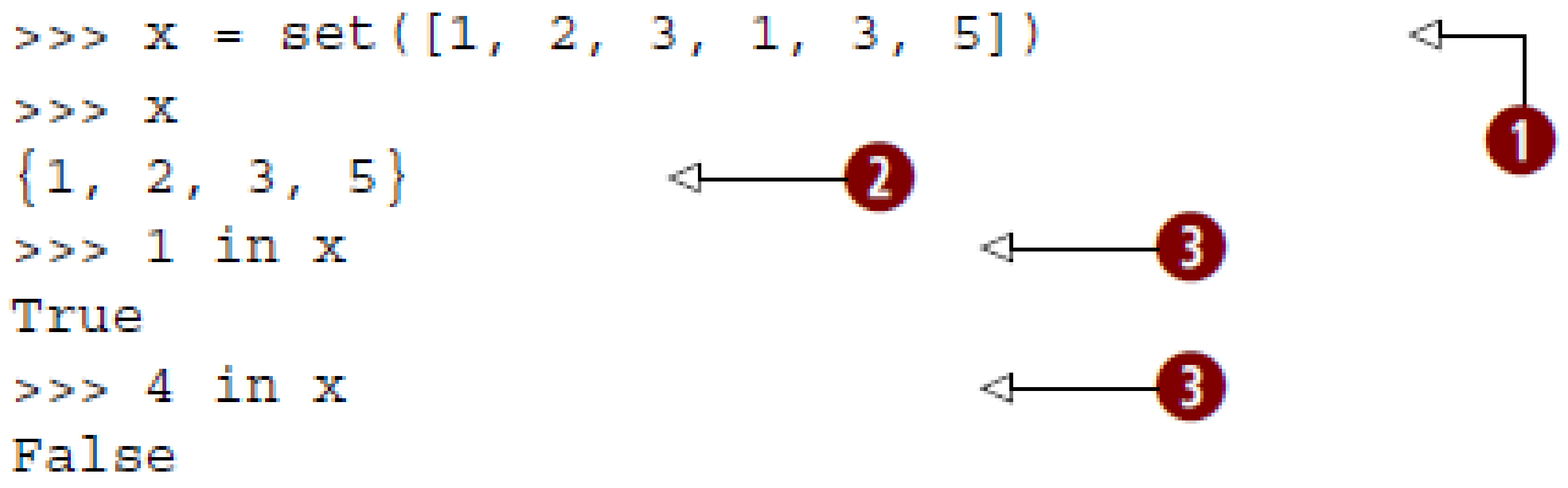
Python 3 – Dictionaries



```
>>> x = {1: "one", 2: "two"}
>>> x["first"] = "one"
>>> x[("Delorme", "Ryan", 1995)] = (1, 2, 3)
>>> list(x.keys())
['first', 2, 1, ('Delorme', 'Ryan', 1995)]
>>> x[1]
'one'
>>> x.get(1, "not available")
'one'
>>> x.get(4, "not available")
'not available'
```

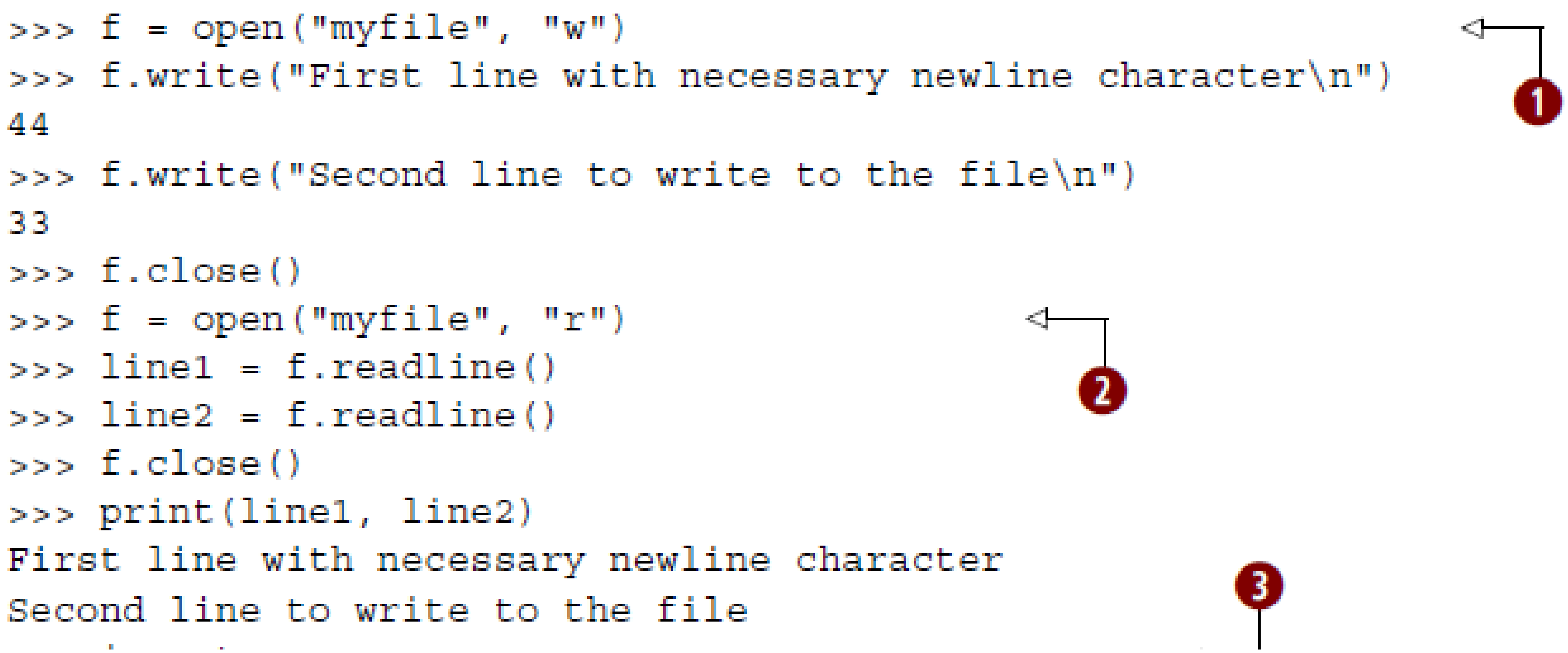
Python 3 – Sets

- A set is an unordered collection of objects



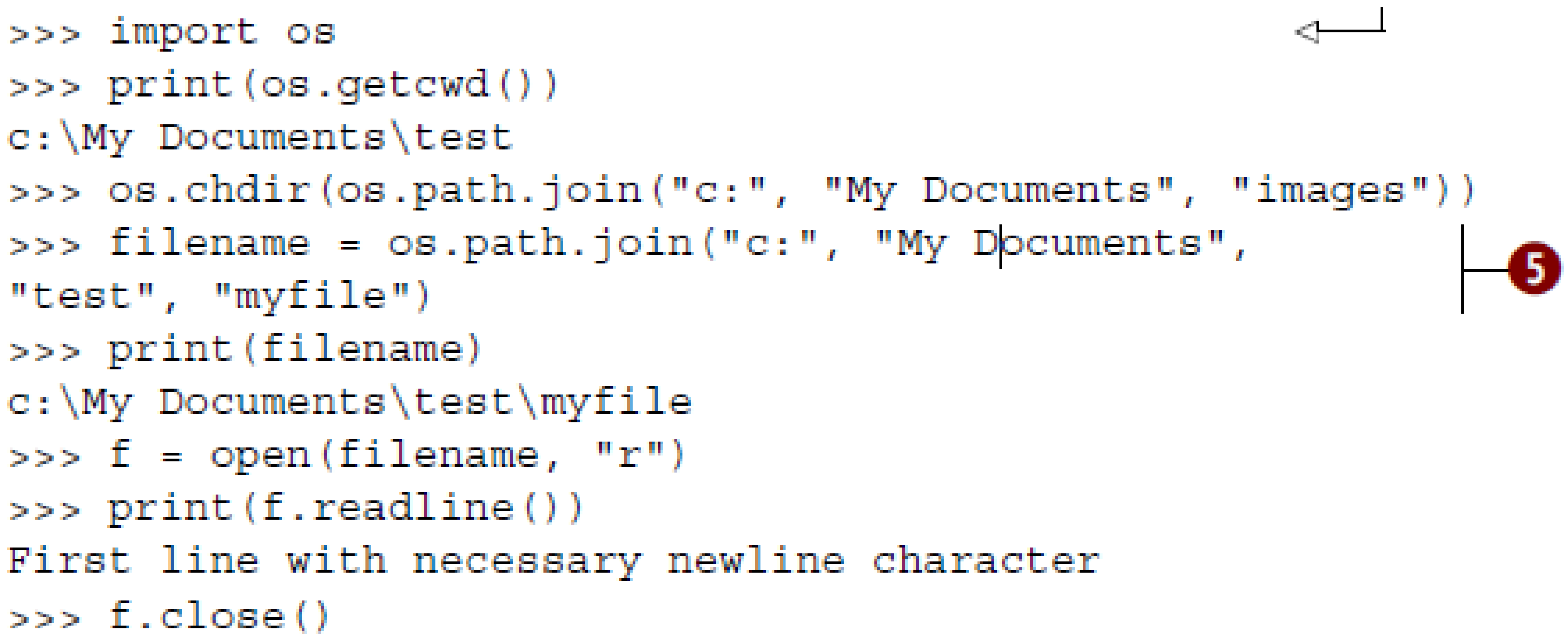
```
>>> x = set([1, 2, 3, 1, 3, 5])
>>> x
{1, 2, 3, 5}
>>> 1 in x
True
>>> 4 in x
False
```

Python 3 – File objects



```
>>> f = open("myfile", "w")
>>> f.write("First line with necessary newline character\n")
44
>>> f.write("Second line to write to the file\n")
33
>>> f.close()
>>> f = open("myfile", "r")
>>> line1 = f.readline()
>>> line2 = f.readline()
>>> f.close()
>>> print(line1, line2)
First line with necessary newline character
Second line to write to the file
```

Python 3 – The os module

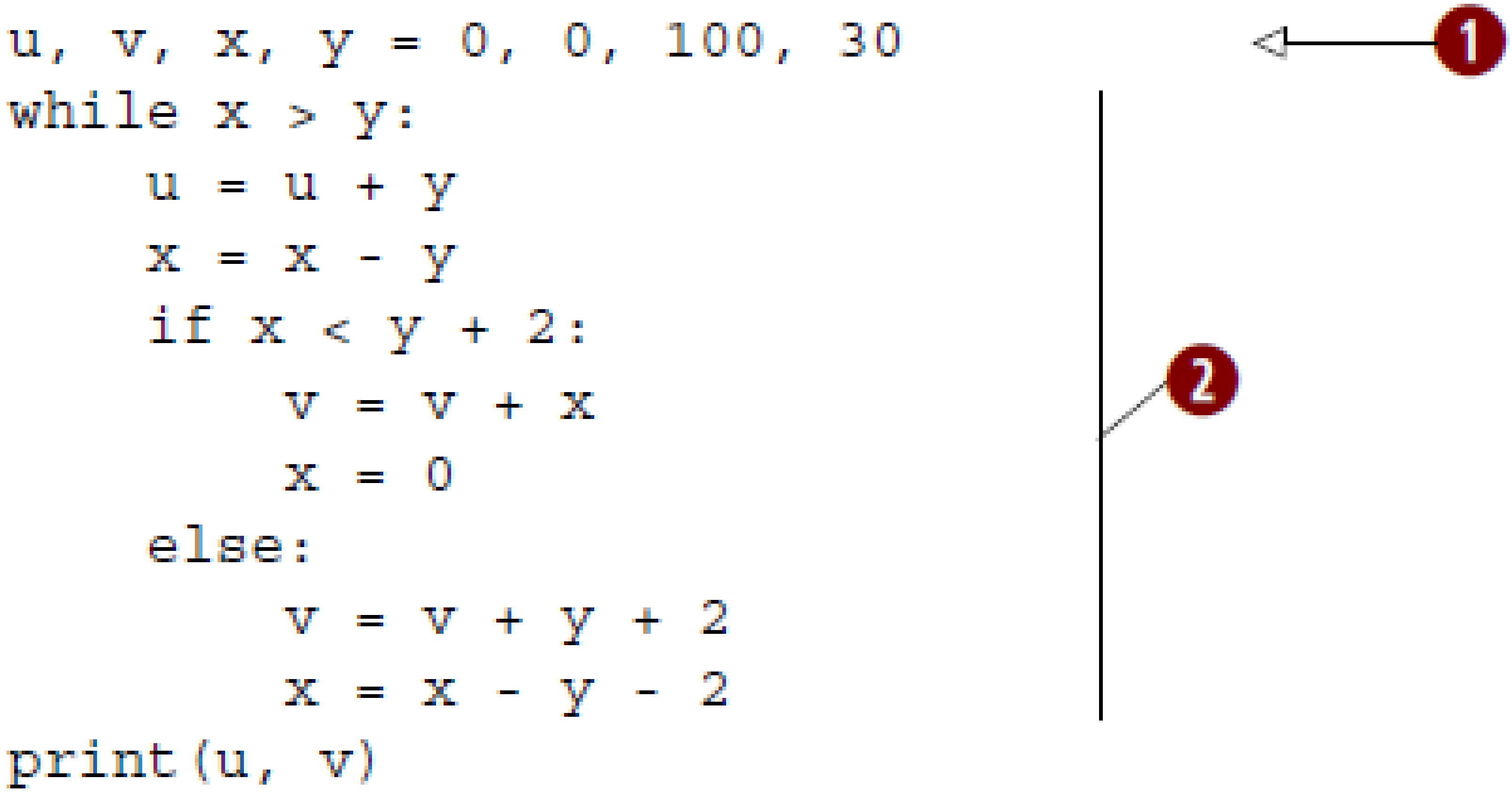


```
>>> import os
>>> print(os.getcwd())
c:\My Documents\test
>>> os.chdir(os.path.join("c:", "My Documents", "images"))
>>> filename = os.path.join("c:", "My Documents",
"test", "myfile")
>>> print(filename)
c:\My Documents\test\myfile
>>> f = open(filename, "r")
>>> print(f.readline())
First line with necessary newline character
>>> f.close()
```

The if-elif-else statement

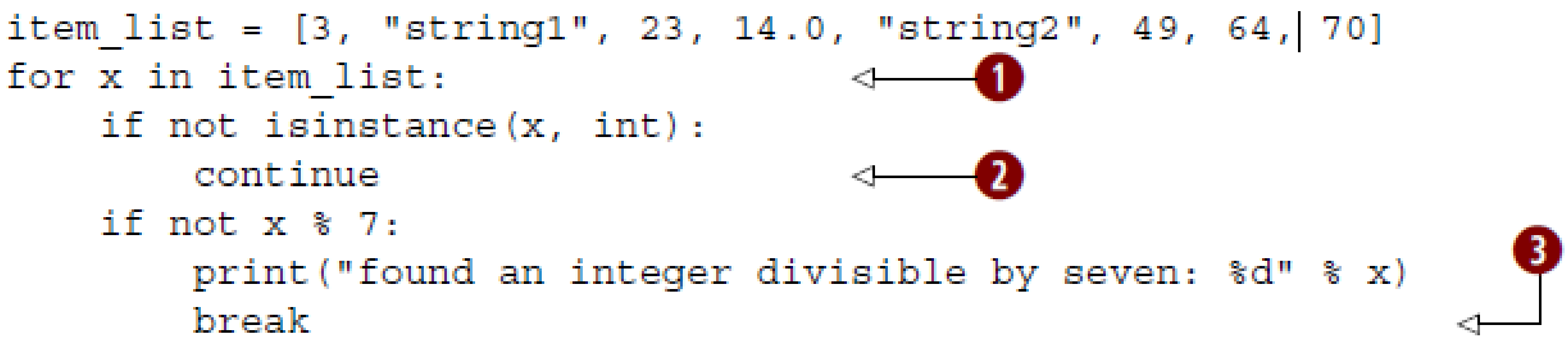
```
x = 5
if x < 5:
    y = -1
    z = 5
elif x > 5:
    y = 1
    z = 11
else:
    y = 0
    z = 10
print(x, y, z)
```

The while loop



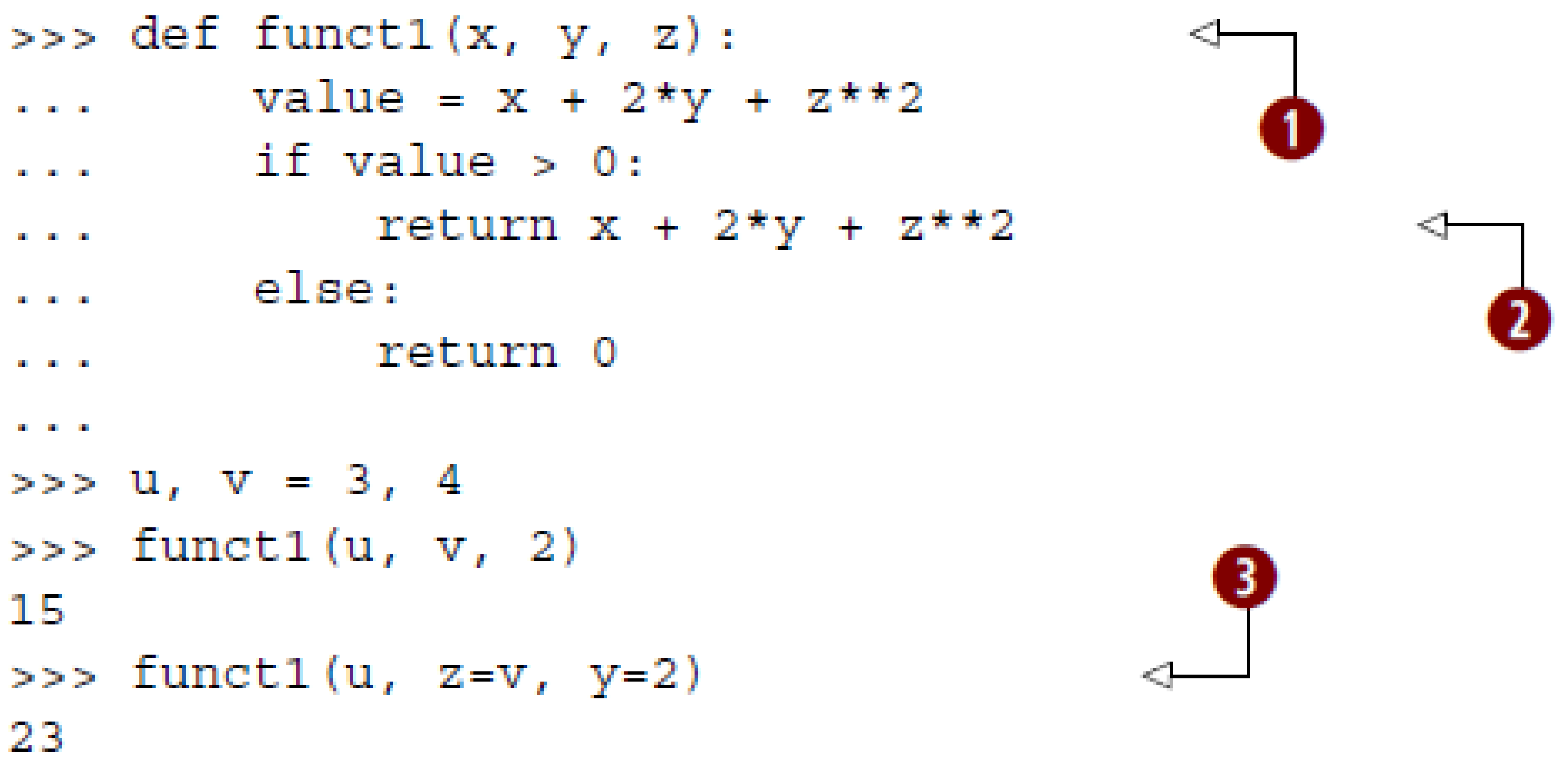
```
u, v, x, y = 0, 0, 100, 30
while x > y:
    u = u + y
    x = x - y
    if x < y + 2:
        v = v + x
        x = 0
    else:
        v = v + y + 2
        x = x - y - 2
print(u, v)
```

The for iterator



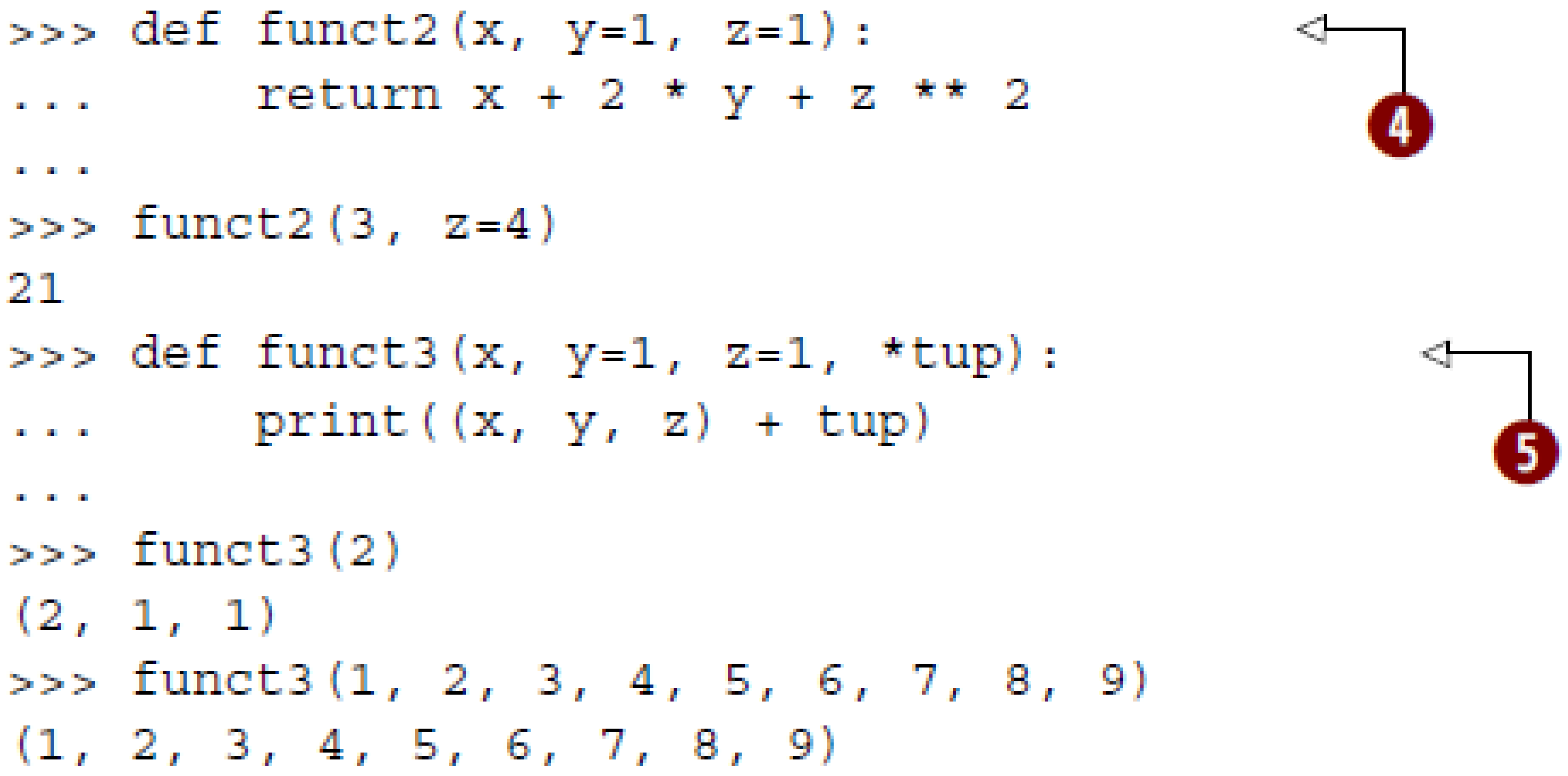
```
item_list = [3, "string1", 23, 14.0, "string2", 49, 64, 70]
for x in item_list:
    if not isinstance(x, int):
        continue
    if not x % 7:
        print("found an integer divisible by seven: %d" % x)
        break
```

Functions



```
>>> def funct1(x, y, z):
...     value = x + 2*y + z**2
...     if value > 0:
...         return x + 2*y + z**2
...     else:
...         return 0
...
>>> u, v = 3, 4
>>> funct1(u, v, 2)
15
>>> funct1(u, z=v, y=2)
23
```

Functions

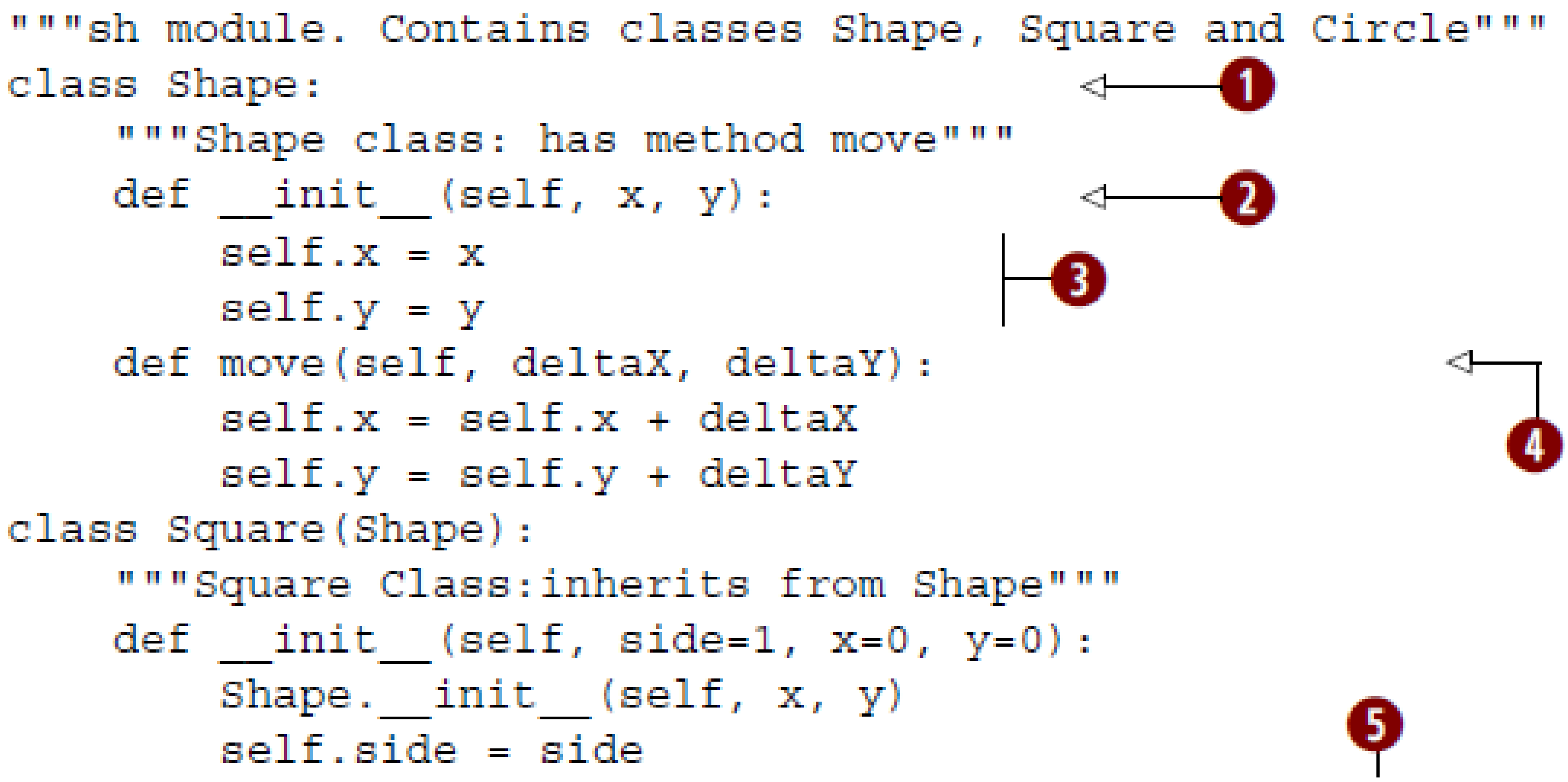


```
>>> def funct2(x, y=1, z=1):
...     return x + 2 * y + z ** 2
...
>>> funct2(3, z=4)
21
>>> def funct3(x, y=1, z=1, *tup):
...     print((x, y, z) + tup)
...
>>> funct3(2)
(2, 1, 1)
>>> funct3(1, 2, 3, 4, 5, 6, 7, 8, 9)
(1, 2, 3, 4, 5, 6, 7, 8, 9)
```

Module creation

```
"""wo module. Contains function: words_occur()"""                        1
# interface functions                              2
def words_occur():
    """words_occur() - count the occurrences of words in a file."""
    # Prompt user for the name of the file to use.
    file_name = input("Enter the name of the file: ")
    # Open the file, read it and store its words in a list.
    f = open(file_name, 'r')
    word_list = f.read().split()                                      3
    f.close()
    # Count the number of occurrences of each word in the file.
    occurs_dict = {}
    for word in word_list:
        # increment the occurrences count for this word
        occurs_dict[word] = occurs_dict.get(word, 0) + 1
    # Print out the results.
    print("File %s has %d words (%d are unique)" \                   4
      % (file_name, len(word_list), len(occurs_dict)))
    print(occurs_dict)
if __name__ == '__main__':                                         5
    words_occur()
```

Object-oriented programming

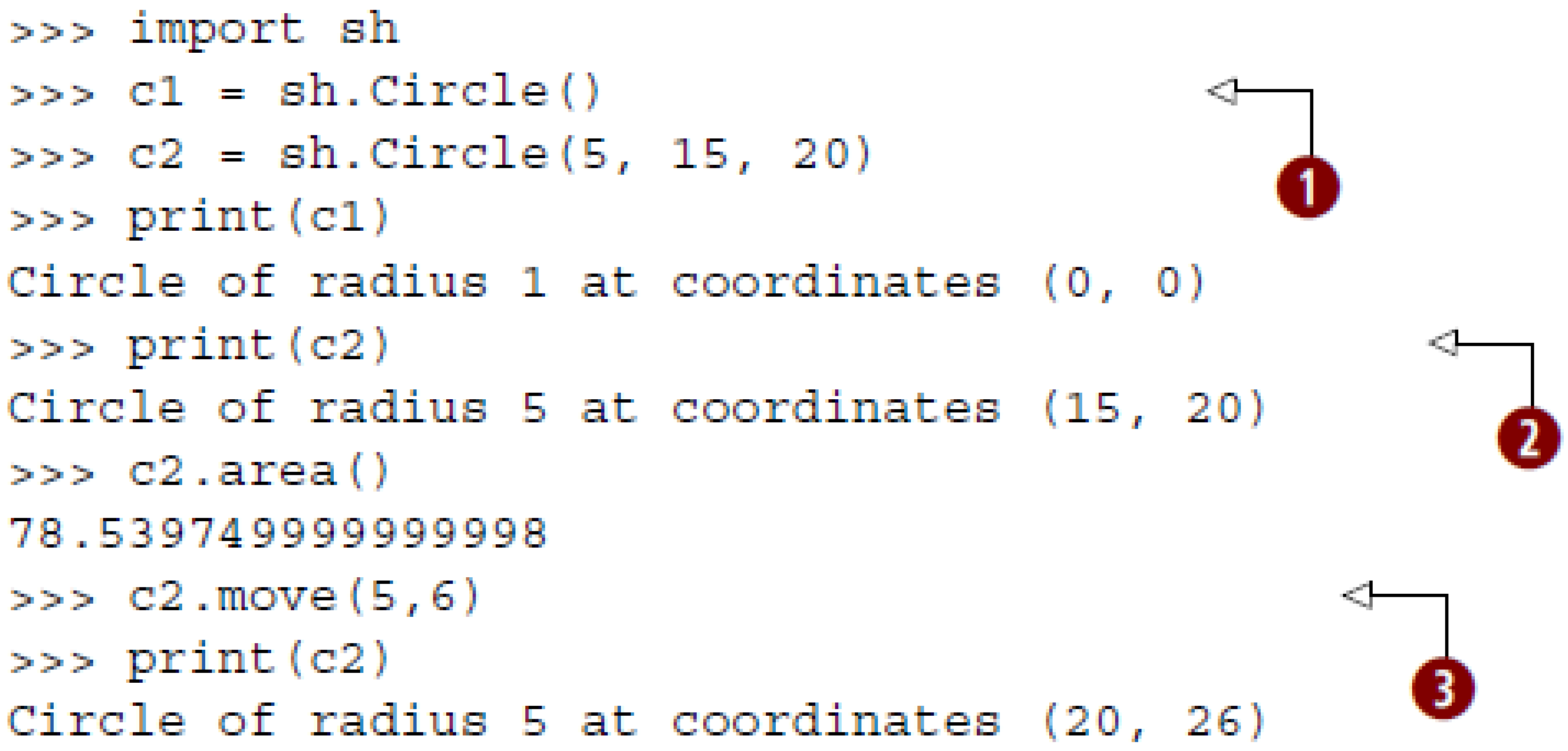


```
"""sh module. Contains classes Shape, Square and Circle"""
class Shape:
    """Shape class: has method move"""
    def __init__(self, x, y):
        self.x = x
        self.y = y
    def move(self, deltaX, deltaY):
        self.x = self.x + deltaX
        self.y = self.y + deltaY
class Square(Shape):
    """Square Class:inherits from Shape"""
    def __init__(self, side=1, x=0, y=0):
        Shape.__init__(self, x, y)
        self.side = side
```

Object-oriented programming

```
class Circle(Shape):
    """Circle Class: inherits from Shape and has method area"""
    pi = 3.14159                                                6
    def __init__(self, r=1, x=0, y=0):
        Shape.__init__(self, x, y)
        self.radius = r                                                    7
    def area(self):
        """Circle area method: returns the area of the circle."""
        return self.radius * self.radius * self.pi
    def __str__(self):
        return "Circle of radius %s at coordinates (%d, %d)"\           8
               % (self.radius, self.x, self.y)
```

Object-oriented programming



```
>>> import sh
>>> c1 = sh.Circle()
>>> c2 = sh.Circle(5, 15, 20)
>>> print(c1)
Circle of radius 1 at coordinates (0, 0)
>>> print(c2)
Circle of radius 5 at coordinates (15, 20)
>>> c2.area()
78.539749999999998
>>> c2.move(5,6)
>>> print(c2)
Circle of radius 5 at coordinates (20, 26)
```

COMPUTATIONAL EFFICIENCY

Efficiency Analysis (Start Here)

- How fast does an algorithm run ?
  - Most algorithms run longer on larger inputs !
- How to relate running time to input size ?
- How to rank / compare algorithms ?
  - If there is more than one available…
- How to estimate running time for larger problem instances ?

Running Time vs. Operations Count

- Running time is not (very) useful for comparing algorithms
  - Speed of particular computers
  - Chosen computer language
  - Quality of programming implementation
  - Compiler optimizations

- Evaluate efficiency in an independent way
  - Count the "basic operations" !!
    - The operations that Contribute the most to overall running time

Input Size

- Relate operations count / running time to input size !!
  - Number of array / matrix / list elements
  - …
- Relate size metric to the main operations of an algorithm
  - Working with individual chars vs. with words
  - Number of bits in binary rep., when checking if n is prime
  - …

Formal Analysis – Pencil and paper

- Understand algorithm behavior
  - Count arithmetic operations / comparisons
  - Find a closed formula !!
  - Identify best, worst and average case situations, if that is the case
- Iterative algorithms
  - Loops : how many iterations ?
  - Set a sum for the basic operation counts
- Recursive algorithms
  - How many recursive calls ?
  - Establish and solve appropriate recurrences

TASK 1 :
ITERATIVE ALGORITHMS

for(i = 1; i <= n; i++) <-> for(i = 0; i < n; i++)

Return value? – Number of iterations?

(Formula of loops witth sum as in code)

Formula of loops always summing 1

```
int f1(int n) {
   int i,r=0;
   for(i = 1; i <= n; i++)
      r += i;
   return r;
}
```

```
int f2(int n) {
  int i,j,r=0;
  for(i = 1; i <= n; i++)
    for(j = 1; j <= n; j++)
      r += 1;
  return r;
}
```

1. $\sum_{i=1}^{n} c = cn$
2. $\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$
3. $\sum_{i=1}^{n} i^2 = \frac{n(n+1)(2n+1)}{6}$
4. $\sum_{i=1}^{n} i^3 = \left[\frac{n(n+1)}{2}\right]^2$

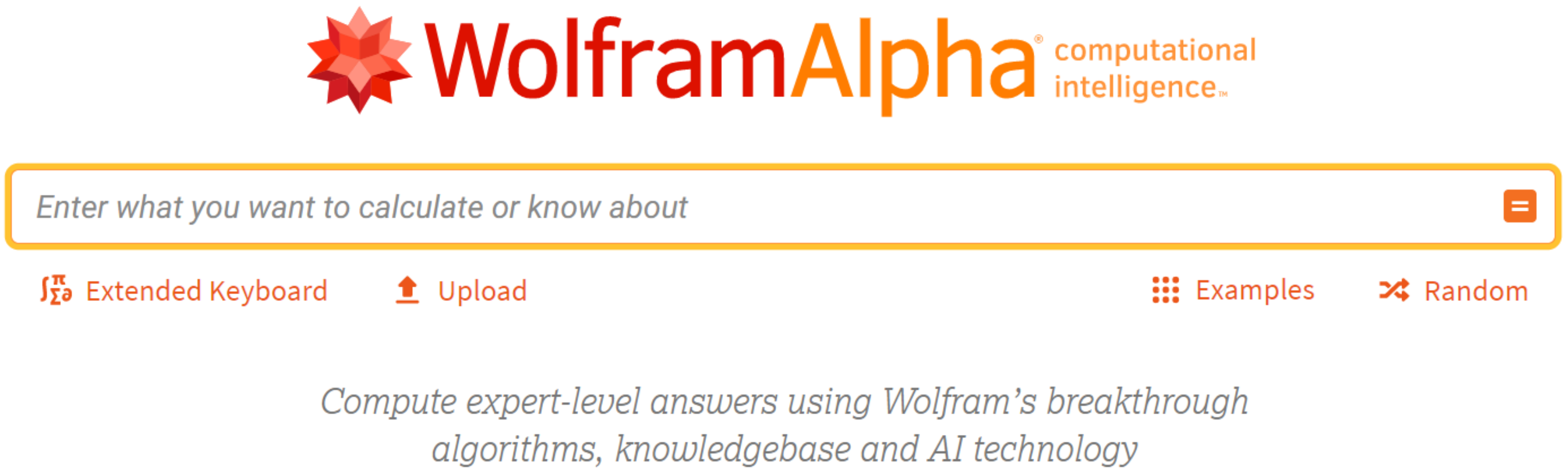
```
int f3(int n) {
  int i,j,r=0;
  for(i = 1; i <= n; i++)
    for(j = i; j <= n; j++)
      r += 1;
  return r;
}
```

```
int f4(int n) {
  int i,j,r=0;
  for(i = 1; i <= n; i++)
    for(j = 1; j <= i; j++)
      r += j;
  return r;
}
```

Tasks – Closed formulas

- Result of each function ?
- Number of times the innermost instruction is executed ?
- Start by writing down summations !
- Use WolframAlpha to check your results !

WolframAlpha


WolframAlpha computational intelligence.

Enter what you want to calculate or know about

Compute expert-level answers using Wolfram's breakthrough algorithms, knowledgebase and AI technology

https://www.wolframalpha.com/

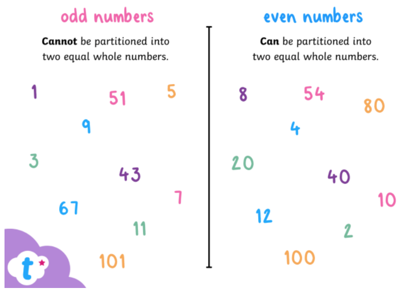
Tasks

- Implement the functions of the previous slide in Python

- Check the correctness of the previously obtained closed formulas

TASK 2 :
RECURSIVE ALGORITHMS

Return value? – Number of calls?



odd numbers
Cannot be partitioned into two equal whole numbers.
1 51 5 9 3 43 7 67 11 101
even numbers
Can be partitioned into two equal whole numbers.
8 54 80 4 20 40 10 12 2 100

```
unsigned int
r1(unsigned int n) {
   if(n == 0) return 0;
   return 1 + r1(n – 1);
}  (returns n basically)
```

```
unsigned int
r2(unsigned int n) {
   if(n == 0) return 0;
   if(n == 1) return 1;
   return n + r2(n – 2);
} sums all positive odd or even numbers until 1 or 0
```

```
unsigned int
r3(unsigned int n) {
  if(n == 0) return 0;
   return 1 + 2 * r3(n – 1);
}  (returns 2^n - 1)
```

```
unsigned int
r4(unsigned int n) {
  if(n == 0) return 0;
   return 1 + r4(n – 1) + r4(n – 1);
} Same as r3 but with two recursive calls
```

(r3 GPT response)

Tasks – Closed formulas

- Result of each function ?
- Number of recursive calls ?
- Start by writing down reccurrences and solving them !
- Use WolframAlpha to check your results !

Tasks

- Implement the functions of the previous slide in Python
- Check the correctness of the previously obtained closed formulas

REFERENCES

References

- V. L. Ceder, *The Quick Python Book*, 2nd Ed., Manning, 2010
  - Chapter 1 + Chapter 2 + Chapter 3
- A. Levitin, *Introduction to the Design and Analysis of Algorithms,* 3rd Ed., Pearson, 2012
  - Chapter 1 + Chapter 2
- T. H. Cormen et al., *Introduction to Algorithms*, 3rd Ed., MIT Press, 2009
  - Chapter 1

Acknowledgment

- The Python overview slides are based on V. L. Ceder's book:
  - V. L. Ceder, *The Quick Python Book*, 2nd Ed., Manning, 2010

- The Python examples are from Chapter 1 and Chapter 3 of that book