BENJAMIN MERLIN BUMPUS, TOM WALLIS, FIONNUALA JOHNSON, SOFIAT OLAOSEBIKAN, ALEXANDRINA PANCHEVA

PWSA 2019 COMPLETE PROBLEMS

PWSAFRICA

Contents

Introduction

This course will introduce you to the Python programming language through a hands-on approach focused on problem-solving. The course covers all of the fundamental constructs of Python and is aimed at beginner programmers. The materials have been chosen because they are fundamental to Python, but also because they are fundamental to many other programming languages and thus, after having completed this course, we expect students to be capable to learn other programming languages on their own. Being able to be a self-reliant learner is a skill and, like all skills, it must be practiced. However, it can also be very time consuming to attempt to learn anything without guidance; for this reason we have compiled this list of problems which we shall help you solve.

A note on the course structure

Throughout the course you will see questions which require you to *read* code and explain what it does. Do not rush through this. Being able to read code (even code which uses constructs which are foreign to you) is a vital skill and it will force you to understand how Python works on a deeper level. When approaching these questions, make sure to ***guess*** an answer before you check it! Constantly trying to predict what the code will do will make you a better programmer.

The first chapter will introduce you to some of the Python fundamentals. At first, this will be a superficial introduction intended to make you familiar with the syntax of the language. In the subsequent chapters you will gradually build on the knowledge by learning how to use these constructs by solving problems of increasing difficulty.

If you are an advanced student (one who already knows how to program in other languages) feel free to skip ahead to whichever section you find challenging. We do, however, still encourage you to complete *all* the reading challenges: you should be able to correctly complete all of them before moving on to more complicated problems. If you finish the material quickly, ask one of the tutors for a larger project (or propose your own) and we shall make sure that you are challenged throughout the course.

A note on attitude

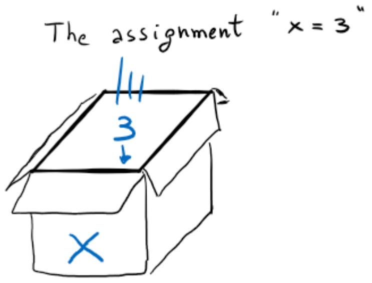
We encourage all students to *ask questions* and *be proactive learners*. Make sure to *engage* and do not, under any circumstances, feel shy about asking questions: it is almost surely the case that many of your fellow students are also confused and it gives us the chance to help you. In general: to make the most of the course, you must put in hard work and, above all, you must

- *try things* and don't be worried if they break — trying, and following the machine's feedback, is an essential part of learning!
- *ask tutors for help*
- *ask questions* (as many as you can!).

Basics

Here we will learn about *variables*, *assignments*, *expressions*, *lists* and *for-loops*. Make sure that you take care in learning these concepts and using the right terminology because the next chapter will focus on problem-solving using the skill you will learn here.

The variable x
X



Learning how to read

What do these snippets of code *do*?

Some of this code might not run properly. If a program has an error, describe the error, and what you think the code was *intended* to do.

Remember: some mistakes will stop a program from running, while others will make a program do the wrong thing!

Reading Challenge 1.

```
a = 5
```

Reading Challenge 2.

```
5 + 6
```

Reading Challenge 3.

```
a = 5 + 6
```

Reading Challenge 4.

```
# a = 5
b = 6
```

Reading Challenge 5.

```
# a = 5
b = 6
c = a + b
```

Reading Challenge 6. Does this work, or does it give us an error? What do you think *might* happen? Test your idea on a computer. Were you correct?

```
a, b, c = 1, 2, 3
```

Reading Challenge 7.

```
height = 5      # 5 meters
weight = 2      # 2 kilograms
gravity = 9.81  # gravitational acceleration is 9.81 m/s^2 on
     Earth
potential_energy = height * weight * gravity
```

Reading Challenge 8. This is an interesting one; what will the `-=` operator do? Is it valid Python?

```
bank_balance = 50
bread = 20
milk = 10
bananas = 10
groceries = bread + milk + bananas
bank_balance -= groceries
```

Reading Challenge 9.

```
bank_balance = 50
bread, milk, bananas = 20, 10, 10
groceries = bread + milk + bananas
bank_balance = bank_balance - groceries
```

Reading Challenge 10. This code contains two new things:

1. "`if`" statements, which let us make *decisions*

Figure 1: Conditionals are like road signs in your code: they allow you to make decisions based on what the state of your code is. The keywords you need to know are `if`, `elif` and `else`.

2. "Indented blocks". In Python, when a set of lines of code is treated in some special way (like being executed only sometimes), we indent the code. You'll see lots of this in Python.

For example, consider the English sentece: "if it is raining, then I will bring an umbrella". Notice that, Even in English, the sentence is split into four parts: "if", the condition, "then" and what happens after the then. If we wanted to write this in Python, we would use indentation to separate the four parts of the sentence (so that the computer knows what's what). This is how our sentence would look like in Python:

```
if (it is raining):
    I will bring an umbrella
```

Notice that, in python, the part of the sentence that comes after the "then" is indented (and we omit the word "then").

Two things to know about indented blocks:

- They *always* begin with a ":" symbol, like you can see above.
- You can have any number of lines inside the indented section. We call this an *indented block*.

Now it's your turn. What is the value of y in the first code snippet and what is the value of x in the second block?

```
x = 5
y = False

if (x > 3):
    y = True
```

```
x = 5
y = False

if (x > 3):
    y = True

if (y == False):
    x = 3
```

Debugging 11. The following code snippets *may* have errors. Your task is to identify which, if any, errors exist in the code. There may be zero, but there may also be more than one!

```
if True
    print "true"
```

Specifically, do the following:

1. Assess the snippet of code and determine what mistakes, if any, exist.

   Work this out without a computer.
2. Once you are certain of your answer, write the program in your computer and check the errors it outputs. Were you correct?
3. Fix the errors in the code, where necessary.

```
if True or False:
    print("true")
```

```
if true or false:
    print("true")
```

```
condition = True or False
if condition:
    print("true")
```

Problem 12. Use Python to help you with computation:

1. Given a right angle triangle with base length 12345 units and height 321 units, find the length of its hypotenuse.
2. Find the roots of the polynomial $4x^2 - 3.59x - 3.14$.

Debugging 13. The following code snippets *may* have errors. Your task is to identify which, if any, errors exist in the code. There may be zero, but there may also be more than one!

Specifically, do the following:

1. Assess the snippet of code and determine what mistakes, if any, exist.

   Work this out without a computer.
2. Once you are certain of your answer, write the program in your computer and check the errors it outputs. Were you correct?
3. Fix the errors in the code, where necessary.

```
print(
    "chunky_chicken"
)
```

```
a = 4 * 5
```

```
a = 4 ** 5
```

```
a == 4 ** 5
```

Reading Challenge 14. Explain what the following code does.

```
bank_balance = 50
shopping_cart = []
bread = 20
milk = 10
bananas = 10
shopping_cart.append(bread)
shopping_cart.append(milk)
shopping_cart.append(bananas)
```

Reading Challenge 15. Does `list()` make a new list, or cause an error?

```
shopping_cart = list()
bread, milk, bananas = 20, 10, 10
shopping_cart.append(bread)
shopping_cart.append(milk)
shopping_cart.append(bananas)
```

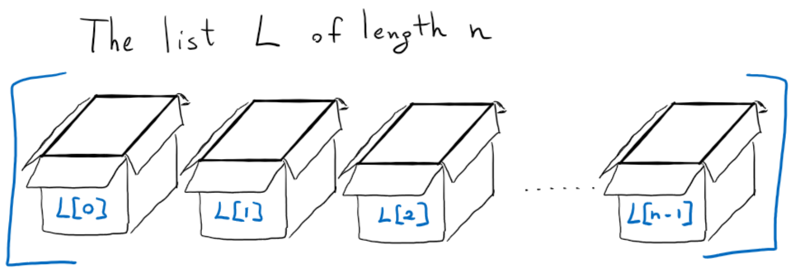


Figure 2: Lists are the fundamental data structure in all programming languages; and they're a natural concept too: a list is just a box full of boxes which are ordered in a line.

Reading Challenge 16. Which of the following is correct?

```
shopping_cart = []
bread, milk, bananas = 20, 10, 10
groceries = [bread]
shopping_cart.append(groceries[1])
```

```
shopping_cart = []
bread, milk, bananas = 20, 10, 10
groceries = [bread]
shopping_cart.append(groceries[0])
```

Reading Challenge 17.

```
bread, milk, bananas = 20, 10, 10
groceries = list()
groceries.append(bread, milk, bananas)
```

Reading Challenge 18.

```
bread, milk, bananas = 20, 10, 10
groceries = [bread, milk, bananas]
bank_balance = 50
cost = groceries[0] + groceries[1] + groceries[2]
bank_balance -= cost
```

Writing

Problem 19.

1. Initialise a list containing the first 10 even numbers.
2. Print out the second element, followed by the last element and the second last element.
    - Index the elements using their exact positions
    - Assign the variable `l` to be the length of the list. Use `l` to find the *last* element in the list.
    - Index the elements using negative indices
3. Print out the length of the list.

Problem 20. Given a list of integers, print the sum of its last two elements and then append it to the end of the list.

Problem 21. Do we actually understand lists?

1. Can a list contain other lists? Test yourself by trying to initialize a list which contains lists of integers (do so in the Python `repl`: just make the list up, it doesn't have to be long).
2. what is the type of the list you initialized?
3. can a list contain lists of lists of lists?
4. is $[4, True]$ a valid list? Is this a good thing or a bad thing? Explain why and discuss this with your friends and with the tutors.
5. if `a` and `b` are two lists, what does `a + b` do? First think about it and *guess* the answer; then try it out with Python.

Problem 22. Using a for-loop, do the following:

- Write out the numbers between 0 and 9
- Write out the numbers between 1 and 10
- Write the numbers backwards between 1 and 10
- Write out all the even numbers between 0 and 50
- Write out all the odd numbers between 0 and 50

- Write all the multiples of 7 less than 100
- Write out all the multiples of 10 counting down from 100 (including 0 and 100)

Debugging 23. The following code snippets *may* have errors. Your task is to identify which, if any, errors exist in the code. There may be zero, but there may also be more than one!

```
for i in range(100):
    print x
```

Specifically, do the following:

1. Assess the snippet of code and determine what mistakes, if any, exist.

   Work this out without a computer.
2. Once you are certain of your answer, write the program in your computer and check the errors it outputs. Were you correct?
3. Fix the errors in the code, where necessary.

Problem 24. Given an integer n, print all the numbers between 0 and $n-1$. For example, if I give you 5, you print 01234 (*note that the numbers are all on the same line*).

Problem 25. Given a list of integers, print the sum of its last two elements and then append it to the end of the list.

Problem 26. Can you find all the even numbers less than 50 using for loop and range only? (tip: range(start, stop, step))

Debugging 27. The following code snippets *may* have errors. Your task is to identify which, if any, errors exist in the code. There may be zero, but there may also be more than one!

```
for i in range():
    print(i)
```

```
for i in range(1, 10):
    print(i)
```

```
for i in range(1, 10, 4, 2):
    i = 5
```

Specifically, do the following:

1. Assess the snippet of code and determine what mistakes, if any, exist.

   Work this out without a computer.
2. Once you are certain of your answer, write the program in your computer and check the errors it outputs. Were you correct?
3. Fix the errors in the code, where necessary.

Becoming fluent in Python

This chapter will consolidate the knowledge you just acquired. When learning a programming language, just like learning a natural language, you have to start by learning a few basic terms and learning some grammar before you can speak. However, after you've done this for while, there is nothing better but to try to speak and write as much as possible. The previous chapter taught you some of Python's *syntax* and *grammar* and introduced some the basic building blocks of the language (you will learn more in this section), now you need to practice "speaking" Python. To do so you need to start using Python for what it is best used: solving problems.

Problem 28. Use a for-loop to find the length of a list.

Problem 29. Given a list of integers, use a for-loop to find its maximum element; then do the same to find its minimum element.

Problem 30. Find the average of a list of integers.

Problem 31. Given a list of integers, can you find the runner-up? For example, given [2, 3, 3, 6, 4, 5], the maximum is 6 and the runner-up is 5.

Problem 32. Suppose you are given the list [1, 1], if you add the last two numbers and append it to the list, the new list will be [1, 1, 2]. If you repeat this addition sequence on the new list three more times, you obtain [1, 1, 2, 3, 5, 8]. Starting with [1, 1], can you print the resulting list if you carry out this addition sequence 50 times? *You've just printed the first 50 elements of the Fibonacci sequence.*

Problem 33. Given two lists L and R, sum their elements in pairs using a for-loop; for example, if $L = [1, 2, 3]$ and $R = [3, 2, 1]$, then their sum should be $[4, 4, 4]$.

Problem 34. Given a square matrix of size $N \times N$ represented as a list of list, print all the elements in its left diagonal. For example, given the 3×3 matrix

$$[[11, 2, 4],\ [4, 5, 6],\ [10, 8, -12]]$$

you are expected to print $11\ 5\ \ -12$ (*note that the delimiter is space*). Can you find the sum of these elements?

Problem 35. Now print all the elements in its right diagonal. Can you find the sum of these elements?

Problem 36. What is the absolute difference between the sum of elements in the left diagonal and the sum of elements in the right diagonal?

Problem 37. Pair up corresponding entries from 2 lists to form a nested list of pairs.

Problem 38. Given a nested list of pairs separate the pair into individual lists.

Problem 39. Given a nested list flatten the list into a single list.

If and Conditionals

Problem 40. Given a list of integers, how many of its elements are positive, negative, and are zeros?

Problem 41. Given a list of integers, how many of its elements are multiples of 3? For example, given the list $[5, 6, 9, 10, 15, 17, 18, 21]$, five of its elements are multiples of 3.

Problem 42. Can you find all the numbers between 1 and 1000 that are multiples of 3?

Problem 43. Can you find all the numbers between 1 and 1000 that are multiples of 3 and 5?

Problem 44. Now print those numbers between 1 and 1000 that are both multiples of 3 and multiples of 5. Can you find the sum of these numbers?

Problem 45. Given an integer N, can you print all the numbers that are factors of N (excluding N)? Can you find the sum of these numbers?

Problem 46. Write a function that prompts the user for a positive number, if the sum of all the factors of that number equals N, print *your number N is a perfect number*; otherwise, print *your number N is not a perfect number*.

Strings

Problem 47. Write out all the characters in a string one per line

Problem 48. Write out all the characters backwards

Problem 49. Write out the first and the last, second and second last. . . .

Problem 50. Write a program which checks if a string is palindromic.

Problem 51. You are given an un-ordered array consisting of integers without any duplicates. You are allowed to swap any two elements. You need to find the minimum number of swaps required to sort the array in ascending order

Problem 52. Two strings are anagrams of each other if the letters of one string can be rearranged to form the other string.

Given two strings, a and b, that may or may not be of the same length, determine the minimum number of character deletions required to make a and b anagrams. Any characters can be deleted from either of the strings.

Problem 53. Two strings are anagrams of each other if the letters of one string can be rearranged to form the other string. Given a string, find the number of pairs of sub-strings of the string that are anagrams of each other.

Problem 54. You are given 3 strings: f, s, and t. t is said to be a shuffle of f and s if it can be formed by interleaving the characters of f and s in a way that maintains the left to right ordering of the characters from each string.

For example, given f = "abc" and s = "def", t = "dabecf" is a valid shuffle since it preserves the character ordering of f and s. So, given these 3 strings write a program that detects whether the third string is a valid shuffle of first and second.

Becoming a Python Poet

Now that you've learned how to write, you need to learn how to write beautifully. This chapter will teach you how to correct your own mistakes (*debugging*) and it will introduce the concept of a *function*. These new ideas will allow you to make your programs neat and tidy and this will, in turn, allow you to solve more complicated and more challenging problems. The chapter will end with a series of more challenging problems which will force you to bring all your learning together before you face the second part of this course.

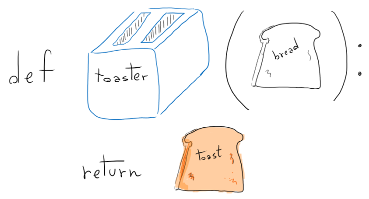


Figure 3: A toaster seen as a *function* which takes as *input* some bread and *returns* as output some toast.

Problem 55. Consider the following code snippet and do the following:

1. write down (on paper) exactly what each line of the code does and state which words are Python keywords
2. why are certain parts of the code indented and others are not?
3. identify (i.e. write down) which lines of code are part of the body of a conditional statement.
4. identify which parts of code define functions, which parts are in a function definition and which parts of the code are not in a function definition
5. explain why one would want to write a function; what is a function and what is it for? (Hint: if we had not defined the function "`cod`", would we have written more or less code?)
6. what does the "`return`" keyword do? What do you think would happen if we removed the third line of code? (take a guess, we don't expect you to know; however, guessing is an important part of learning!)
7. what differences are there between the variable "`eleven`" and the variable "`twelve`"?

.

```
def addthree(x,y,z):
  theResult = x+y+z
  return theResult

def cod(animal):
    if(animal == "dog"):
        return "your␣dog␣is␣a␣cat"
    elif(animal =="cat"):
        return "your␣cat␣is␣a␣dog"
    else: return "all␣animals␣are␣either␣cats␣or␣dogs"

cat = "elephant"
eleven = 12
twelve = "11"
if(eleven < 11):
    cat = cod("sheep")
else:
    cat = cod("dog")
x = addThree(1,2,3)
```

Debugging 56. The following code snippets *may* have errors. Your task is to identify which, if any, errors exist in the code. There may be zero, but there may also be more than one!

Specifically, do the following:

1. Assess the snippet of code and determine what mistakes, if any, exist.

   Work this out without a computer.
2. Once you are certain of your answer, write the program in your computer and check the errors it outputs. Were you correct?
3. Fix the errors in the code, where necessary.

```
def myfunction(a list):
    return a list[0]
```

```
y = 10
def myfunction(x):
    return y / 5
```

Problem 57. The following function takes as input two parameters: an integer a and a list of integers b. Your tasks are:

1. for each line in the following function, write down on a piece of paper **exactly** what it will do as it is executed. Use the correct terminology for the components you refer to in the line. An explanation of the first line is given, as an example,

2. write down what the final output of the program is when we run $\texttt{foo}(10, [1, 4, 2])$.

```
def foo(a,b):   # define a function ‘foo’ with 2 params.: ‘a’
     & ‘b’
  ell = len(b)
  for i in range(ell):
    b[i] = b[i] - a
    a = a - b[i]
  return b
```

Problem 58. Write a function that reads two integers representing a month and day and prints the season for that month and day.

Debugging 59. The following code snippets *may* have errors. Your task is to identify which, if any, errors exist in the code. There may be zero, but there may also be more than one!

```
def myfunction:
    return 5
```

Specifically, do the following:

1. Assess the snippet of code and determine what mistakes, if any, exist.
   Work this out without a computer.
2. Once you are certain of your answer, write the program in your computer and check the errors it outputs. Were you correct?
3. Fix the errors in the code, where necessary.

Problem 60. At normal atmospheric pressure, water changes state to a solid at $0C$ or below and a gas at $100C$ or above. It remains a liquid at any other temperature.

Write a function that will return “solid”, “liquid” or “gas” to the user depending on the temperature they enter.

Problem 61. Write a function, called `middle`, which returns the element in the middle of the list.

If the list has an even number of elements, you may decide what to do.

Problem 62. Write a function, called `first`, which takes as arguments a list and an integer and which returns the first `n` elements of `elems`[1].

[1] Consider: what happens if `n` is bigger than `len(elems)`?

Problem 63. Given two sides of a triangle write a function to work out the size of the angle.

Problem 64. Write two functions called ‘`addLists`’ and ‘`weirdListAdding`’ which both take a **list of lists of integers** as input. Do so in such a way that:

1. 'addLists' uses the function you defined in problem 33 to add all the elements of the input list.
2. 'weirdListAdding' does the same as 'addLists', except, before it adds the elements of the input list, it reverses all of the elements of the input list (you can reverse lists using the function you defined in problem **??**)

Problem 65. Using a `for` loop and a `list`, write a function called `mymap` which takes two arguments — a function, and a list — and returns that function applied to all of the elements in the list. For example:

```
my_list = [1,2,3]
def double(x):
    return x*2

# We want mymap to be written so that:
output = mymap( [1,2,3], double )  # returns the list [2, 4,
    6]
```

Problem 66. Sometimes we have big lists with lots of elements, but we only want certain ones. We can write a test to see whether a given element is one we want. For example, if we wanted to filter our list so it only contained even numbers, we could first write the following function to test a single element of the list:

```
def isEven(number):
    if number % 2 == 0:
        return True
    else:
        return False
```

...and then another function to apply the function to our list[2].

[2] Fun fact! A function like isEven which takes an argument and returns either `True` or `False` is called a `Predicate`.

Your challenge is to write this function, called `myfilter`, which takes two arguments: a list to filter, and a function which will return `True` if the item is to be kept, and `False` if it is to be thrown out. For example:

```
def isEven(number):
    return number % 2 == 0

myfilter([3, 2, 5, 4, 6, 7, 1, 2], isEven).  # Should return
    [2, 4, 6, 2].
```

Problem 67. Write a function to read in a string and return a string without punctuation.

Problem 68. Write a function which, given a string as input, returns a list of the words it contains.

Problem 69. Write a function to remove any words beginning with the letter *b* from a list.

Reading Challenge 70. This is a complicated one. Assume `should_apply` and `to_apply` are functions, and vars is a list. Can you work out what this function is intended to do?

```
def f(should_apply, to_apply, vars):
    output = list()
    for item in vars:
        if not should_apply(item):
            output.append(item)
        else:
            output.append(to_apply(item))
    return output
```

Review: Harder Questions

Problem 71.

1. Write a function which removes duplicate elements from a list.
2. Write a function that returns the intersection of two lists (without duplicates)
3. Write a function which takes as input two lists and removes from the first list all the elements which it shares with the second.
4. Write a function which takes as input two lists and returns a list containing their union (without duplicates).
5. Write a function that returns the symmetric difference of two lists.

Problem 72. Write a game of hangman. Here are some steps:

1. Take a string and print out a dash for each character

2. Take a string and a letter and print out every occurrence (in place) of the letter in the string and a dash in the place of every other character

3. Take a string and print out a dash for each character. Ask the user to input a letter if the letter appears in the string print out the sting with every occurrence (in place) of the letter and a dash in the place of every other character.

Problem 73. In cryptography, a Caesar cipher[3] is one of the simplest and most widely known encryption techniques. It is a type of substitution cipher in which each letter in the original text — referred to as the "plaintext" — is replaced by a letter some fixed number of positions down the alphabet.

[3] Also known as Caesar's cipher, the shift cipher, Caesar's code or Caesar shift

Write two functions: `encryptCaesar` and `decryptCaesar` which, given an integer n, encrypt and decrypt (respectively) an input string.

You can read more about Ceaser ciphers at this page.

Problem 74. In English, the frequencies of the occurrence of the letters of the alphabet in known words are as follows (in alphabetical order)

$$[8.167, 1.492, 2.782, 4.253, 12.702, 2.228, 2.015, 6.094, 6.966, 0.153,\\ 0.772, 4.025, 2.406, 6.749, 7.507, 1.929, 0.095, 5.987, 6.327, 9.056,\\ 2.758, 0.978, 2.360, 0.150, 1.974, 0.074].$$

Use this information to crack the encryption of a string where the encryption is known to be done using a Caesar Cypher.

Fundamentals for the Advanced User

Dictionaries

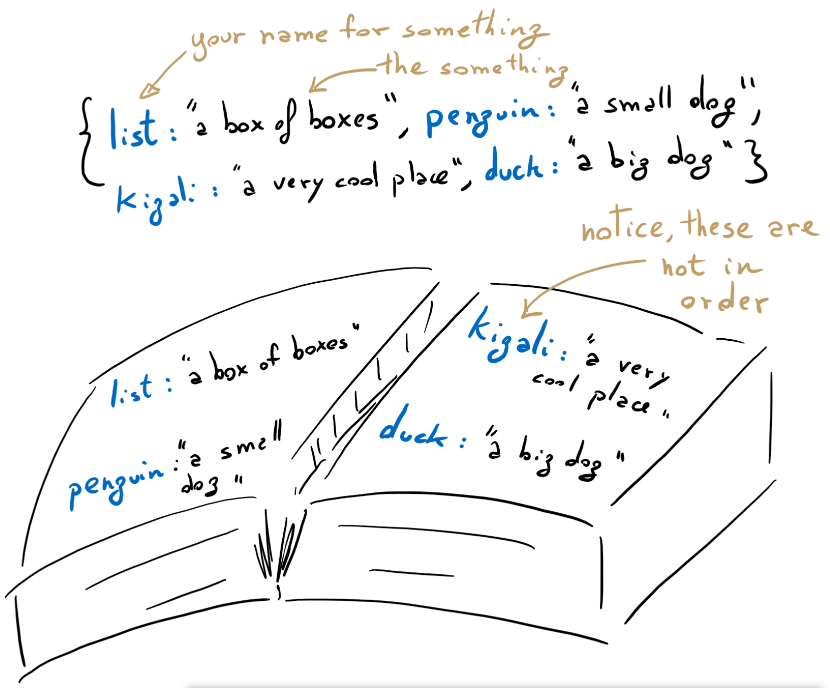

your name for something
the something
{ list : "a box of boxes", penguin : "a small dog",
kigali : "a very cool place", duck : "a big dog" }
notice, these are not in order
list : "a box of boxes"
penguin : "a small dog"
kigali : "a very cool place"
duck : "a big dog"

Figure 4: Two perspectives on a dictionary: in Python, in "real" life.
Oil on canvas, circa 1492, artist unknown provided to the gallery of PWSA by an anonymous donation.

Dictionaries are usually described by computer scientists as a data structure containing "key-value pairs". This means that dictionaries contain combinations of "keys", which are unique, and "values", which the dictionary remembers as being related to the key in some way. For example, a phone book could be considered a "dictionary" of people's names to people's phone numbers: the names are unique "keys", and by selecting a key, we can find the "value" related to it; here, a phone number.

In a way, dictionaries are similar to lists: a list relates a number to a value (where the number is an index in the list). So, the list:

```
["chunky", "bacon", "is", "tasty"]
```

relates the index, `2`, to the value `"is"`. This is more or less equivalent to the dictionary:

`{0:"chunky", 1:"bacon", 2:"is", 3:"tasty"}`

Dictionaries can accept almost any type of key and value. One exception is that dictionaries are not valid keys. They are valid values, for example!

Reading Challenge 75. Describe (in words, written on paper or in a text file) what kind of dictionary the following code snippets define.

```
phonebook = dict()
phonebook["John␣Paul"] = 07924316548
phonebook["Fionualla␣Johnson"] = 07439532325
```

```
shopping_list = {}
shopping_list["tinned␣food"] = ["tomatoes", "beans", "tuna"]
shopping_list["vegetables"] = ["yam", "potato", "tomatoes"]
```

```
doubled_numbers = {2:4, 3:6, 8:16, 25:50}
```

```
hotel = {}
floor_1 = {101: "Mr.␣Akeju", 103: "Dr.␣Praise␣Adeimo"}
floor_2 = {202: "Dr.␣Kitty␣Meeks"}
floor_3 = {} # This floor is empty!

hotel["floor␣1"] = floor_1
hotel["floor␣2"] = floor_2
hotel["floor␣3"] = floor_3

hotel["floor␣4"] = {401: "Dr.␣Tim␣Storer", 402: "Joe␣
    Gallagher", 405: "Nicola␣Sturgeon"}
```

Problem 76. Write a dictionary which takes strings as their keys

Problem 77. Write a function that takes the dictionary and prints it out, the keyword followed by the numbers in the list(not in the list)

Problem 78. Write a function of your choosing which takes and returns a number (such as `def double(x): return x*2`).

Write another function which checks to see whether the input has been seen before. If it has not, use the function to calculate the result and *save the calculation in a dictionary*, where the key is the input and the value is the result. If it has been seen already, look up the result in the dictionary. For the double function given as an example, you should be able to build the relevant dictionary from the reading problems at the beginning of this section.

This technique is usually referred to as Dynamic Programming.

Once you have solved this problem, consider other functions you could have written. Are there situations where this technique could make your programs more elegant or quicker? What are those situations? Discuss with a tutor if you can get their attention.

Thinking with Data Structures

Problem 79. Represent a family tree as a data structure — you may choose which — and then do the following:

- Write a function to return a person's parents
- Write a function to return a person's children
- Write a function to return a person's siblings
- Write a function to return a person's aunt's and uncles
- Write a function to return a person's cousins
- Write a function to return a person's is a direct ancestor of another person.

Randomness

Problem 80. Write a function called '`crazyCoin`' which, given an integer n as input, returns "`Heads`" n% of the time and "`Tails`" the other times.

Problem 81. Write a function which

- generates a random positive integer (call it n) less than 100
- generates a list (call it '`crazyCoinTosses`') containing n 'coin tosses' (where each 'coin toss' is obtained by using the function defined in problem 80 with input n).

Problem 82. Write a function which randomly shuffles a list.

Problem 83. Given a list $L = [c_1, ..., c_n]$ of n distinct characters, write a program which generates a random string s such that every character of s is an element of L and such that character c_i occurs i times in s.

Problem 84. Write a function called 'rps' which takes as input a string x and then generates a random string of either 'rock", "paper" and "scissors" and then: if x is one of 'rock", "paper" or "scissors", then 'rps' returns "you" if x wins the rock-paper-scissors game and "me" otherwise; if x is **not** one of 'rock", "paper" or "scissors", then 'rps' returns "I don't think you know how to play this game...".

Problem 85. Prompt the user for an input within a range. Check that the input is within the required range. If it is not keep prompting the user until it is acceptable.

Problem 86. Create a list of 10 random numbers without repetition.

Problem 87. Write two functions called `toDecimal` and `toBinary` such that the first converts decimal numbers to binary and the second converts binary numbers to decimal.

Problem 88. Guessing game

File Input Output

Problem 89. Write a function that reads in a text from a file

Data Science Fundamentals

Numpy

Problem 90. Snail dataset (same as last year) can be a good introduction to indexing, slicing, calculating average, max, etc.

Matplotlib

Problem 91. Choosing the most appropriate visualisation for a range of datasets.

Problem 92. (Whisky Dataset): Heatmap, plot distilleries on the map of Scotland. Find a whiskey most similar to another?

Problem 93. Interactive heatmap of geographical data using gmplot (possibly accidents data)

Computational Thinking Problems

Problem 94. Let's say we're standing in a building, with 100 floors. I give you a pack of strange eggs, with thick shells. When dropped from a certain — but unknown — height, an egg will break. To test this, we could drop one egg from each floor, and see at which floor they begin to break. However, this could waste as many as 100 eggs.

Without the use of a computer (for example, using paper and pen), solve the following.

- If I give you *only* 50 *eggs*, can you still find the exact floor? What method can you use to do this?
- What is the fewest number of eggs you can drop to find the *exact* floor they begin breaking on?
- For `f` floors of a building, what is the minimum number of eggs you can use?