How to approach Python.

Introduction.

You want to learn how to program and have just read that Python is one of the easiest programming languages to learn. So on you go and 5 minutes later you have made your first working code and 15 minutes later you have constructed your first game. That is not how it works out in the real world and you may already have discovered that it is true. It takes quite a lot more to write a piece of code than that.

So how do you approach Python or, for that matter, any other programming language? Well, look at Python as a tool. Conceptualising Python as a tool at first glance may seem a simple and odd approach, but it will get you to understand what programming in Python is all about.

So let us dwell at the term tool or tools. In a daily manner of speech a tool could be a hammer, screwdriver or a saw. Depending on what you want to achieve you pick the appropriate tool and do the work. It is obvious that the hammer is best suited to punch a nail into a wall. However, removing the nail with a hammer may not be the best solution, but it can be done. Either you tab on the nail from different directions to loosen up the nail, so that you can remove the nail without a brute force or, you choose to demolish the wall which, eventually, will remove the nail from the wall. The latter may not bring a smile up on your family's faces. If you had chosen to use the screwdriver to dig a small hole around the nail, it would have been at better solution. So back to Python.

Python is a tool, or even better a tool box. You can reach inside the Python tool box and choose any tool you like to solve your problem. The tricky part is to choose the right tool.

Python comes at installation with "batteries included". This means that the basic tools, and even more advanced tools, are available to you from the very beginning. Your job is to investigate the tools and get an idea on how to use them – this is the point where your imagination becomes a valued asset. So imaginary speaking, it can be compared to the situation where you want to play with Lego. You buy a big box of Lego bricks, opens it and picks the Lego blocks you will need to build a small house. When you are skilled in building small houses you may wish to take it a step further and build small houses with entrance doors and from there maybe add some windows. Maybe even a few rooms and a second floor. If you from the very start want to build a house with a third floor, it may be very difficult to do that, unless you know how to build the second floor first and at least have a floor plan.

So your approach to Python, before you actually know anything about the programming language, should be: Python is a toolbox and you have to learn how to use the tools inside it.

The fundamentals.

Being new to programming it may be very difficult to see where to begin, but as for most learning processes, beginning with the fundamental parts is a good start. Programming is not all about writing code, it is a process of making a row of selections. Basically this means, that the code makes some kind of evaluations on certain conditions and, depending on the conditions, a choice is taken.

In Python, depending on the version of Python, you have to be very familiar on how it handles integers. For example 5 / 2 = 2, which to us is wrong because we all know that the result is 2.5. Well, in Python this is right. This is the way Python 2.7 handles division with integers. But if we expect to get 2.5 as a result and pass it on, we are getting the wrong result. Analysing this example merely suggest, that Python tell us, that 2 can be divided up in 5, 2 whole times, leaving a remainder 1, which Python do not inform us about. Enlightened with that information, we now know that Python on division do not implicitly return a decimal number, as expected.

The above example suggest, that a part of the fundamental knowledge must be concentrated on how Python handles numbers, since these are responsible for choices in a majority f all evaluations in code.

It may be rather childish that you have to work your way through the operator signs +, -, / and *, but if you want to learn how to write code that works, there is now short cuts.
An other part of Python is the excellent features on handling strings and here strings are "immutable". Hey – this is not a course in biology. Well you are right, but strings are after all immutable, that is: You can not change a string. But you can iterate through each letter in a string and even have these printed.
One of Pythons great features is also the backbone in Python: The ability to handle list, dictionaries and tuples, gives Python an immense power. Both list and dictionaries are mutable, whereas tuples are immutable. So a highly in-depth study on how to handle list, dictionaries and tuples is a must. Questions like: How do I access a number in a list?, how to delete a number or string from a list?, how do I make a copy of a list?, how do I create an empty list?, how to insert values in a dictionary? and loads of other questions have to be answered and, indeed tried out, before you are ready to take the next step into programming.

Writing code.
Being new to Python, you will very fast learn to write some code. For some it only takes a few hours and others a couple of days. Within days of programming you will find out, that most code is written in functions and that functions interact with each other. Nice – you have now reached a point where you are on the edge to be a full certified programmer or? Have you ever considered that you might have to read your own code in 6 month? Can you read and understand it? Well, this depends on how you wrote it.
Many new to programming leaves out the description on their functions. This often leads to troublesome work later on. So do your self a great favour: Learn how to write the functions description, the type contract and testing examples. In writing these few and short instructions, you quite often stumbles over the coding solution to your function. An other great benefit by doing this is, that other programmers immediately can see what your function actually does.

Code complexity – is that really important?
Code complexity – why bother? My computer is fast and can do anything. Well, that may not be the correct answer. Let us take a good look at a simple problem.
Guess the Number is a very popular game and can easily be coded. The idea is to guess a number between 0 and 100, where 100 is not included. In this version you pick the number and the computer suggest a number. If it is your number – computer wins, if not – you tell the computer whether the guess was too high or too low.
There are two different approaches to solve this problem: The iterative way and the sneaky way. So let us see which of these two methods is the best and we assume that the numbers is in a ordered list.
The iterative way:
Iteratively we could have the computer to compare all numbers from 0 to 99 with the secret number. The best scenario is when the secret number is 0. However the worst case is to run through all numbers until 99 is reached. If that is the case it is going to take a while before the game ends but, eventually the computer will guess the secret number.
In this situation the worst case scenario is what we have to expect, and henceforth the maximum of runs before the secret number is found is 100. The complexity of this code is straight forward and is equal to the length of the numbers: len(n).
The sneaky way:
We can do better. Since the list of numbers is ordered we could choose an other method on how to pick the numbers and, at the same time reduce the number of times we have to make a guess. The method is called bisection or binary search. Using binary search we immediately starts with a qualified guess. Since bisection is about continually splitting the search area in halves, the first number we will try as a guess is 50. If the number is to high, we have eliminated all numbers larger than or equal to 50. If we manually bisect the interval from 0 to 100, we will see that this can be

done at most 7 times. So implementing this solution raises the complexity of our code and drastically reduces the number of guesses.
Code complexity is measured after the Big O notation. Studying the iterative solution you will find that the complexity is equal to the length of the string – in other words linear. The examination of the code for a binary search result in a logarithmic complexity – to be more precise log2.

What do we get? Well, if the range of numbers is close to 0, i.e. range(0, 10) we would have to make 4 guesses in stead of 10, before we get the right result. Here we could have chosen to stay with the iterative solution. But as the range increases, the complex code is to be preferred. Here is some numbers to reflect what we are dealing against.

Log2(100) = 7
Log2(1000) = 10
log2(10000) = 14
log2(1000000) = 20

In other words, if you want to find information on a specified person, expressed by the persons social security number, you will have to search for example 100,000,000 individual security numbers. That is .100 million times through the iterative search before the information is found. A binary search only needs 27 loops before the result is found.

Handling modules.
Python comes with built-in modules. To use these you have to import these and for beginners are often introduced to modules like math, random and timedate. There are however other modules in the standard installation that is worth the time to investigate. Common for the modules is, that you at any time can find help on specified functions. Here the dir and help command are very helpful and it is a very good idea to learn these two functions by heart.
Python have the advantage, that you can create your own modules. There are 2 ways of accessing a module. If you have created a file area.py you can access the functions within the file from an other Python file. As long as the 2 files are in the same library, you can directly write: import area.
If the files not in the same area you can not access the functions within the area.py file. Here you need to know, that the area.py file must be copied to the Python27\Lib or you have to write the path to the file each time you want to access it. Developing your own modules is one of Pythons great features.

Graphical User Interface – yes man, it's a must or?
GUI's can be very useful, but for a start beginners should concentrate on the basics. There is a good reason for this: GUI are used to illustrate something like movements in a game, visualising particle collisions and reflections or visualise a database. So if visualisation is not desperately needed, try to make your code without. After all, Python delivers all answers in the promt and other results can easily be printed to a file.
There are however some beginners to Python that do a parallel study on the GUI's and they do face some problems: They are fighting 2 battles at the same time. In most situations you will loose this approach to Python, mainly because you fight with handling the basic elements in Python and at the same time are struggling to understand how the GUI's like PyGame work. Even Napoleon knew that fighting on two opposite fronts eventually will lead to a defeat – but not every time.

Remember Python is fun.

Holmes