

nostarch.com/pfk

For bulk orders, please contact us at [sales@nostarch.com.](mailto:sales@nostarch.com)

|  |  |  |  |
| --- | --- | --- | --- |
| Teacher: |  | Date/Period: |  |
| Subject: | Python Programming | Class: |  |
| Topic: | #1 - Getting Started | Duration: | Up to 50 mins |

|  |  |
| --- | --- |
| Objectives: | * Student can install the latest Python3 distribution * Recognises the difference between the Python console and the Shell (IDLE) - why use one over the other? * Create a simple program. Run it from the command line, and from the Shell (open the file and run it, all from within in the Shell environment) * Use the Shell (or Console) as a simple calculator * Student can describe what a program is, and what a programming language is |
| Materials: | * Python 3 installer (or link to download the installer) * A short Python program for the student to run (as a suggestion, one of the turtle examples from Chapter 11: <http://jasonrbriggs.com/python-for-kids/code.html#Chapter-11>) |
| Activities: | 1. Download the latest version of Python and install (depending on the environment/school lab you may want to shortcut this process by predownloading, making the installer accessible from the school network, and so on). [5 - 10 mins] 2. Start up the Python Console, explain what the prompt is. Start up the Python Shell and compare. Try out basic print statements in both. [5 - 10 mins] 3. Open a Python program and run from within the Shell. [5 mins] 4. Create a new window in the Shell, enter a simple program (hello world), save and run. [5 mins] 5. Discussion: programming languages - instructions to the computer; human-readable versus computer-readable, the difference between scripting languages (such as Python) versus compiled languages (such as C or C++) which would be used to create most of the programs the student might use on a day-to-day basis. [10 mins] 6. Open the Shell again, try entering some basic calculations and seeing the result (demonstrate the different operators: + - \* /). Use of brackets and how they effect the result of a calculation. [5 - 10 mins] |
| Notes: | 1. In terms of installation, depending on your lab, either get the student to download from the Python website, or from a local cache, or from the school website - but the process should, at the very least, be reproducible at home. 2. Take careful note of the installation instructions on pages 5 - 10, specifically adding a shortcut to run the Shell in ‘No Subprocess’ mode on Mac and Windows. Necessary, because otherwise the student will experience errors later. |
| References: | Python for Kids, Chapters 1 & 2 (up to page 18) |

|  |  |  |  |
| --- | --- | --- | --- |
| Teacher: |  | Date/Period: |  |
| Subject: | Python Programming | Class: |  |
| Topic: | #2 - ‘Things’ in Python | Duration: | Up to 60 mins |

|  |  |
| --- | --- |
| Objectives: | * Understand the use of variables to ‘store’ things - difference between a slot in memory used to hold the actual value, and a variable being a label ‘pointing’ at the value * Understand the difference between a number and a string * Understand the use of lists and maps * Understand the difference between a tuple and a list * Use a tuple with a string containing placeholders |
| Activities: | 1. Brief review of prior lesson. Opening the Shell and running a program [5 mins] 2. Student creates a program to store a number in a variable and then print out the number. Discussion: what would we use variables for? Why do programs need variables? [5 - 10 mins] 3. What about storing a sentence in a variable? Have the student try to create a variable containing a silly sentence. After error message is displayed, discuss the difference between strings and numbers. Create different variables with strings and numbers [5 - 10 mins] 4. Discussion: Are there other things we might want to store in variables? Introduce lists and maps. [5 mins] 5. Create variables with simple lists of things (student’s favourite items, for example). Removing items from a list by their index. Discussion: if we stored a list of items in a variable, how would we remove an item from the list? Think about a simple instruction - someone gives you a list of things on paper and then says, “I don’t want the 3rd item in the list” - how do you remove that item if the list is a string? You have to recreate the string (which doesn’t make as much sense if you think about the computer having to do it). [10 - 15 mins] 6. Create a tuple. Try to remove an element from the tuple. Discussion: why use a tuple over a list? (Tuples are faster because the values can’t change) [10 mins] 7. Create a string with a couple of placeholders (%s). Use a tuple for the values when printing out the string. Discussion: what is Python doing here? Why are placeholders useful? [10 mins] |
| Notes: |  |
| References: | Python for Kids, Chapters 2 & 3 |

|  |  |  |  |
| --- | --- | --- | --- |
| Teacher: |  | Date/Period: |  |
| Subject: | Python Programming | Class: |  |
| Topic: | #3 - Drawing with the Turtle | Duration: | Up to 50 mins |

|  |  |
| --- | --- |
| Objectives: | * Understand how to import the turtle module and can draw simple shapes * Basic understanding of what a module is (as a unit of code) |
| Activities: | 1. Brief review of prior lesson: Variables, numbers and strings, lists and maps. [5 mins] 2. Open the Shell and import the turtle module. Discussion: what is a module? [5 mins] 3. Draw a line with the turtle. Discussion: a line is made up of pixels - what is a pixel? (possible additional activity: get a magnifying glass and see if students can see the edge of a pixel on the screen) [5 - 10 mins] 4. Turn the turtle 90 degrees and draw another line. Discussion: do the students know what degrees are? If not, use a clock to illustrate the major degrees (45, 90, 135, etc). Try turning the turtle 90 degrees right and 90 degrees left. Get the students to draw a zig-zag line. [10 mins] 5. Try drawing a square with different sizes. Reset and then try drawing a square on an angle. Try using the up and down commands to start and stop drawing. Generally free experimentation at this point. [15 - 20 mins] |
| Notes: | 1. If your students haven’t setup Python according to the instructions in Chapter 1, this is where they may experience problems - such as the Shell seeming to hang when trying the use the turtle module. 2. When discussing ‘modules’, focus on the fact that they are like a small program that you can use inside your own programs. 3. Depending on the level of the students it may be a good idea to cover a lesson on degrees in parallel to this lesson (such as: http://www.homeschoolmath.net/teaching/g/measure\_angles.php). 4. If the student wants to start their drawings again, two options: use the reset function, or close the window and start again. |
| References: | Python for Kids, Chapter 4 |

|  |  |  |  |
| --- | --- | --- | --- |
| Teacher: |  | Date/Period: |  |
| Subject: | Python Programming | Class: |  |
| Topic: | #4 - Control Statements | Duration: | Up to 60 mins |

|  |  |
| --- | --- |
| Objectives: | * Understand what blocks of code are in Python * Understand what an if-statement is, and how to use it * Basic understanding of functions |
| Activities: | 1. Brief review of prior lesson: the turtle module, and a module being a unit of code that you can use in your programs [5 mins] 2. Discussion: what a block code in Python? How Python identifies a block of code. Why do we need blocks of code [5 mins] 3. Discussion: if-statements are used to control which block of code to run. How to create an if-statement. [5 mins] 4. Students open the Shell, create a new editor window, and then type in a variable ‘age’ with their age as the value. Next they create a simple if-statement which prints one statement if the variable is greater than a specified value, otherwise prints another statement (go for the silliest messages possible here). Discussion: what do the students think their code is going to do when they run it, and why? [5 - 10 mins] 5. Once they’ve run it and seen one message displayed, get them to change the code, so that the other statement is printed. Do they understand why the code works the way it does? [5 mins] 6. Discussion: now break down the code describing the keywords (if/else), the condition (for example, age > 11), and which of the statements form the two code blocks. [5 mins] 7. Get the students to alter the conditions. Cover each conditional symbol (equals ==, != not equal, greater than >, less than <, greater than or equal to >=, less than or equal to <=) [5 - 10 mins] 8. Discussion: what might you use an if-statement for (given the examples so far, this is a valid question)? Brief description of a function being a small unit of code which (usually) returns a value. Give the students an example of using the ‘int’ and ‘input’ functions so that someone can type their age and it will be saved as a variable. Get them to change their code to use the functions, and change the print messages, run the code, then swap computers with another student. [15 mins] |
| Notes: | * Worth mentioning to the student that they’ve been using functions since the second lesson (printing out a number using the print function, for example) |
| References: | Python for Kids, Chapter 5 |

|  |  |  |  |
| --- | --- | --- | --- |
| Teacher: |  | Date/Period: |  |
| Subject: | Python Programming | Class: |  |
| Topic: | #5 - Control Statements | Duration: | Up to 55 mins |

|  |  |
| --- | --- |
| Objectives: | * Understand how to repeat code (loops) * Using the range and list functions (basics of) |
| Activities: | 1. Discussion: Brief review of prior lesson: if-statements, blocks of code, functions [5 mins] 2. Discussion: why would you want to repeat a block of code in a program? (as a starting point for the discussion, how about a playable character on the screen - if you push the right arrow the character needs to walk to the right - how do you make him move across the screen without something like repetition). [5 mins] 3. Have the student open the shell and get them to try range(10) - then try list(range(10)). Brief discussion: what is happening when they run the first function (perhaps go into iterators), versus running the second function? Variation: now try list(range(5, 10)) [5 mins] 4. Now have the student enter a simple for-loop:   for x in range(10):  print('Hello there')  Discussion: what is this code doing? [10 mins] 5. Now use a placeholder in a string with a loop to print the value. Get the students to try the following code:   for x in range(5, 10):  print('Hello there %s' % x)  Discussion: what is this code doing? [10 - 15 mins] 6. Demonstrate that the for loop will also work with lists of things (rather than just functions) by getting the student to try out the following code:  some\_words = ['amazed', 'flabbergasted', 'flummoxed'] for x in some\_words:  print(x)  Discussion: what is this code doing? Is there a difference to the previous code where we used range? [10 - 15 mins] |
| Notes: | n/a |
| References: | Python for Kids, Chapters 6 |

|  |  |  |  |
| --- | --- | --- | --- |
| Teacher: |  | Date/Period: |  |
| Subject: | Python Programming | Class: |  |
| Topic: | #6 - Code Re-use | Duration: | Up to 55 mins |

|  |  |
| --- | --- |
| Objectives: | * Understand how to create basic functions * Using the time module, using the sys module |
| Activities: | 1. Discussion: Brief review of prior lesson: for-loops, the list and range functions [5 mins] 2. Discussion: Python has built-in functions (such as print, list and range), but you can also create your own functions. Why is this useful? (functions are like mini-programs/you can re-use code in different places in your programs/and so on) [5 mins] 3. Have the students open the shell and create a simple function to print their name and then get them to run it. e.g.:  >>> def simplefunc():  print('My name is Maximillian Shufflebottom')  Discussion: talk through parts of a function (the def keyword, the function name, the body) [5 - 10 mins] 4. Change the function to accept a parameter name, and then print the name in the function body:  >>> def simplefunc(name):   print('My name is %s' % name)  Discussion: talk about functions taking parameters. [5 - 10 mins] 5. Now create a new function so that it returns a value. The savings function in Chapter 7 might be a good example:  >>> def savings(pocket\_money, paper\_route, spending):  return pocket\_money + paper\_route – spending  >>> print(savings(10, 10, 5)) 15  Discussion: What’s happening here? (worth mentioning that return is a keyword, similar to def). [5 - 10 mins] 6. Discussion: The students have already used the turtle module, but Python has a lot of different modules. Discuss relationship between functions and modules (as groups of functions and variables) - all ways of code re-use. [5 mins] 7. Have the student import the time module and print the result of the asctime function. Then import the sys module and use the readline function of the stdin object (sys.stdin.readline()). Discussion: check student’s understanding of what’s happening here (focus on the fact that stdin is a variable in the sys module - don’t go into objects at this point) [10 mins] |
| Notes: | * The time and sys modules aren’t the most interesting, but using them helps plant the idea that there’s more to modules than just the turtle. |
| References: | Python for Kids, Chapter 7 |