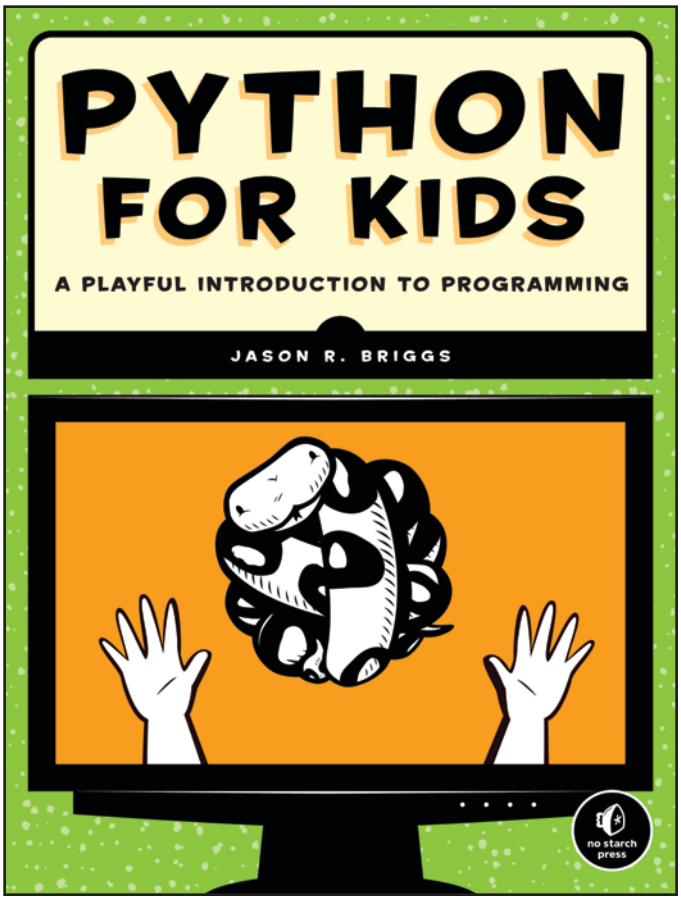

NOSTARCH.COM/PFK

For bulk orders, please contact us at
sales@nostarch.com.

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

Objectives:	• Install the latest Python3 distribution. • Recognize the difference between the Python console and the shell (IDLE) and know when to use one over the other. • Create a simple program. Run it from the command line and from within the shell. • Use the shell (or console) as a simple calculator. • 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 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 and install the latest version of Python. (Depending on the environment, you may want to shortcut this process by predownloading or making the installer accessible from the school network.) [5–10 min.] 2. Start up the Python console, and explain what the prompt is. Start up the Python shell and compare. Try out basic print statements in both. [5–10 min.] 3. Open a Python program and run it from within the shell. [5 min.] 4. Create a new window in the shell, and enter a simple program (for example, Hello World). Save and run the program. [5 min.] 5. Discussion: How are programming languages instructions to the computer? What does it mean for something to be human-readable versus computer-readable? What's the difference between scripting languages (such as Python) and compiled languages (such as C or C++)? Which would be used to create most of the programs seen on a day-to-day basis? [10 min.] 6. Open the shell again. Try entering some basic calculations and seeing the results. Demonstrate the different operators (add `+`, subtract `-`, multiply `*`, and divide `/`), and show how the use of brackets affects the result of a calculation. [5–10 min.]

Notes:	• Depending on your lab environment, have students download from the Python website, from a local cache, or from the school website. At the very least, the process should be reproducible at home. • Take careful note of the installation instructions on pages 5–10, specifically those on adding a shortcut to run the shell from the desktop on Mac and Windows, to avoid experiencing errors later.
References:	*Python for Kids*, Chapters 1 & 2 (up to page 19)

Teacher:		Date/Period:	
Subject:	Python Programming	Class:	
Topic:	#2 Storing Things in Python	Duration:	Up to 60 min.

Objectives:	• Understand how a variable stores things. Know the difference between a slot in memory used to hold an actual value and a variable used as a label to point at a value. • Understand the difference between a number and a string. • Understand the purpose of lists and maps. • Understand the difference between a tuple and a list. • Use a tuple with a string containing placeholders.
Activities:	1. Briefly review the prior lesson: opening the shell and running a program. [5 min.] 2. Create a program to store a number in a variable, and then print out the number. Discussion: What do you use variables for? Why do programs need variables? [5-10 min.] 3. Try to create a variable containing a silly sentence. After the error message is displayed, discuss the difference between strings and numbers. Create different variables with strings and numbers. [5–10 min.] 4. Discussion: Are there other things you might want to store in variables? Introduce lists and maps. [5 min.] 5. Create variables with simple lists of things (students' favorite items, for example). Remove items from a list by their index. Discussion: If you stored a list of items in a variable, how would you remove a single item from the list? How do you remove that item if the list is a string? (You have to recreate the string.) [10–15 min.] 6. Create a tuple. Try to remove an element from the tuple. Discussion: Why use a tuple over a list? (Tuples are faster because their values can't change.) [10 min.] 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 min.]
Notes:	N/A
References:	*Python for Kids*, Chapters 2 & 3

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

Objectives:	• Understand how to import the `turtle` module and draw simple shapes. • Understand what a module is (as a unit of code).
Activities:	1. Briefly review the prior lesson: variables, numbers and strings, lists and maps, and tuples. [5 min.] 2. Open the shell and import the `turtle` module. Discussion: What is a module? (Describe it as a small program that you can use inside your own programs.) [5 min.] 3. Draw a line with the turtle. Discussion: A line is made up of pixels—what is a pixel? Optional activity: Get a magnifying glass and have students find the edge of a pixel on the screen. [5–10 min.] 4. Turn the turtle 90 degrees and draw another line. Try turning the turtle 90 degrees to the right and 90 degrees to the left. Draw a zig-zag line. Discussion: What do degrees measure? Optional activity: Use a clock to illustrate the major degrees (45, 90, 135, etc.). [10 min.] 5. Try drawing a square at different sizes. Reset and then try drawing a square on an angle. Try using the `up` and `down` commands to start and stop drawing. Offer a period of free experimentation. [15–20 min.]
Notes:	• If your students haven't set up Python according to the instructions in Chapter 1, they may experience problems when using the `turtle` module, such as the shell seeming to hang. • Depending on the math level of the students, it may be a good idea to cover a lesson on degrees in parallel to this lesson (for an example, see *http://www.homeschoolmath.net/teaching/g/measure_angles.php/*). • If the students want to start over with their drawings, there are 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 min.

Objectives:	• Understand what blocks of code are in Python. • Understand what an `if`-statement is and how to use it. • Understand what a function is.
Activities:	1. Briefly review the prior lesson: the `turtle` module and a module as a unit of code that you can use in your programs. [5 min.] 2. Discussion: What is a block of code in Python? How does Python identify a block of code? Why do you need blocks of code? [5 min.] 3. Discussion: `if` statements are used to control which block of code to run. How do you create an `if` statement? [5 min.] 4. Open the shell, create a new editor window, and then type a variable `age` with the student's age as the value. Create a simple `if` statement that prints one statement if the variable is greater than a specified value and another if it is not. (Get creative with silly messages here.) Discussion: What do you think the code is going to do when you run it? Why? [5–10 min.] 5. Run the code once to display the message. Then change the code so that the other statement is printed. Review how the code works. [5 min.] 6. Discussion: Break down the code, describing the keywords (`if` and `else`), the condition (for example, `age > 11`), and the statements forming the two code blocks. [5 min.] 7. Have students alter the conditions. Cover each conditional symbol (equal `==`, not equal `!=`, greater than `>`, less than `<`, greater than or equal to `>=`, and less than or equal to `<=`). [5–10 min.] 8. Discussion: What might you use an `if` statement for (given the examples so far)? Briefly describe a function as a small unit of code that usually returns a value. Give the students an example of using the `int` and `input` functions. Have students type their age and save it as a variable. Update the code to use the `int` and `input` functions, and change the print messages. Run the new code, and then have students swap computers with each other. [15 min.]
Notes:	• Remind students 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 min.

Objectives:	• Understand how to repeat code (loops). • Use basic `range` and `list` functions.

Activities:	1. Discussion: Briefly review the prior lesson: `if` statements, blocks of code, and functions. [5 min.]
	2. Discussion: Why would you want to repeat a block of code in a program? (As a starting point for the discussion, consider a character's movements in a game—if you push the right arrow, the character needs to walk to the right. How do you make him move across the screen without repetition?). [5 min.]
	3. Open the shell and run `range(10)` Then run `list(range(10))`. Brief discussion: What happens when you run the first function (perhaps discuss iterators) versus when you run the second function? Variation: Run `list(range(5, 10))`. [5 min.]
	4. Enter a simple `for` loop, for example:
	`for x in range(10):`
	`  print('Hello there')`
	Discussion: What is this code doing? [10 min]
	5. Now use a placeholder (`%`) in a string with a loop to print the value. Try the following code:
	`for x in range(5, 10):`
	`  print('Hello there %s' % x)`
	Discussion: What is this code doing? [10–15 min.]
	6. Demonstrate that the `for` loop will also work with lists of things (rather than just functions) by entering 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 from the previous code where we used `range`? [10–15 min.]

Notes:	N/A

References:	*Python for Kids*, Chapter 6

Teacher:		Date/Period:	
Subject:	Python Programming	Class:	
Topic:	#6 - Code Reuse	Duration:	Up to 55 min.

Objectives:	• Understand how to create basic functions. • Use the `time` module and the `sys` module.

Activities:	1. Discussion: Briefly review the prior lesson: `for` loops and the `list` and `range` functions. [5 min.]
	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 reuse code in different places in your programs.) [5 min.]
	3. Open the shell and create a simple function for students to print their names, for example:
	`>>> def simplefunc():`
	`  print('My name is Maximillian Shufflebottom')`
	Run the function. Discussion: Talk through the parts of a function (the `def` keyword, the function name, and the function body). [5–10 min.]
	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 min.]
	5. Now create a new function that returns a value, for example, the `savings` function in Chapter 7:
	`>>> def savings(pocket_money, paper_route, spending):`
	`  return pocket_money + paper_route – spending`
	`>>> print(savings(10, 10, 5))`
	`15`
	Discussion: What's happening here? (Mention that `return` is a keyword, similar to `def`.) [5–10 min.]
	6. Discussion: The students have already used the `turtle` module, but Python has a lot of different modules. Discuss the relationship between functions and modules (as groups of functions and variables)Show that they are both ways to reuse code. [5 min.]

	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 the students' 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 min.]
Notes:	• The `time` and `sys` modules aren't the most interesting, but using them demonstrates that there are other modules besides the turtle.
References:	*Python for Kids*, Chapter 7