This is the old page - New page is Introduction to Python

[Teacher's Edition] Introduction to Python

Suggested Class Time: approximately 80 minutes

This document serves as a teacher's edition to the Introduction to Python for Students. It contains answers to the Introduction to Python Worksheet, as well as notes to teachers and related CA standards.

What is Python?

Python is a programming language used to create *programs* that tell computers, step-by-step, how to solve problems. Programs are behind all of the computer applications you use every day. Software engineers at Google write programs that power the applications you are familiar with such as Search, Gmail, and Maps. As you will soon learn, you too can write programs to solve interesting problems.

Why use Python in the Classroom?

A computer program gives students the opportunity to directly apply the algorithms they learn in class and provides them with a tangible reason for using variables rather than specific numbers in math. The following are suggested uses for how to incorporate Python programming into any lesson:

- Project a completed program and ask students to analyze the algorithm.
- Project a program and run it, allowing students to see the algorithm in action.
- Add/delete a variable, change a formula, or input different numbers into a program to demonstrate how a change in the algorithm or values affects the result.
- Have students work individually or in pairs to modify and complete partially written programs.
- Once familiar with Python, students can begin to write their own programs from scratch to answer questions related to nearly any topic taught in class.

Python has two different modes, the interpreter mode and the editor mode. The ***Python IDLE interpreter*** works similarly to a calculator in which we can type in math problems and Python computes the answer. The ***Python IDLE editor*** is where we can write our own programs, analogous to adding more features to a calculator so that it better suites our needs.

For instructions on how to install Python or to learn more about Python in general, go to:

- Python Download and Install Instructions
- Python - The Official Site
- Think Python: How to Think Like a Computer Scientist

Teacher's Note: *Exercises 1 - 6 below are aligned with the following CA Standards:*

- ***Grade 6, Algebra and Functions, Standard 1.4****: Solve problems manually by using the correct order of operations or by using a scientific calculator.*
- ***Grade 7, Algebra and Functions, Standard 1.2****: Use the correct order of operations to evaluate algebraic expressions such as 3(2x + 2.5).*

Parentheses in Python *(suggested time: 10 - 15 minutes)*

Parentheses can change the result of an arithmetic expression. In the ***Python IDLE interpreter***, enter the following expressions to understand how parentheses affect the order of operations.

Exercise 1	Exercise 2	Exercise 3
1a) `>>>6+10/2` Result: 11	2a) `>>>2+3*5` Result: 17	3a) `>>>(6+(8/4)-2)*3` Result: 18
1b) `>>>(6+10)/2` Result: 8	2b) `>>>(2+3)*5` Result: 25	3b) `>>>6+(8/4)-2*3` Result: 2

Read through the following steps and calculate the expressions.

Several important tips to keep in mind include:

- Be sure to include all the parentheses you need to perform the correct order of operations.
- Be sure to close any parentheses that you open.
- Use the '*' symbol for multiplication, e.g. 4*(3+1), not 4(3+1).
- Use the '/' symbol for division.

Exercise 4	Exercise 5
4a) **Step 1:** Divide 70 by 5 **Step 2:** Multiply the result by 3 **Step 3:** Subtract 12 from the product	5a) **Step 1:** Multiply 5 by 3 **Step 2:** Add 6 to the product **Step 3:** Multiply the sum by 4 **Step 4:** Subtract 8 from the product **Step 5:** Divide the result by 4
4b) Write the expression described above: (70/5)*3-12	5b) Write the expression described above: ((5*3+6)*4-8)/4
Type the expression into the interpreter. If correct, Python should tell you that the answer is 30.	Type the expression into the interpreter. If correct, Python should tell you that the answer is 19.

Mathematical Notation in Python *(suggested time: 5 minutes)*

Python uses several symbols for mathematical operations that vary from what we usually see in math class. Test the examples below while looking for *patterns* that indicate the meaning of each symbol.

> **Teacher's Note:** *Familiarity with the nuances of Python's mathematical notation is necessary for any student who will be using Python in math class. Encouraging students to work in pairs and discuss the results of Exercises 7 - 10 will help them to remember what these symbols mean when they appear again in the programs later in this lesson.*

Exercise 6

For each of the operations below, enter the three examples into the ***Python IDLE interpreter*** to figure out what each operation does. Use the last column to comment on what you have learned about the operation.

Operation	Example 1	Example 2	Example 3	What does this operation do?
6a) **	>>>7**2 Result: 49	>>>8**2 Result: 64	>>>2**4 Result: 16	Raises the first number to the power of the second number
6b) %	>>>10%3 Result: 1	>>>24%5 Result: 4	>>>18%6 Result: 0	Calculates the remainder when the first number is divided by the second number
6c) /	>>>19/5 Result: 3	>>>21/4 Result: 5	>>>21/4.0 Result: 5.25	Divides the first number by the second number, however Python cuts off the decimal portion of the quotient if neither the dividend nor divisor contains a decimal point.

Equality in Python *(suggested time: 10 minutes)*

Python has three different symbols to represent different types of equality. In the exercises below, test out the '=', '==', and '!=' symbols looking for *patterns* that indicate what each one means.

Exercise 7
Enter the expressions below, starting with 7a), and record the results.

Note:

- *Be sure to complete 7a) before entering in the other expressions; Python cannot evaluate an expression involving a variable until that variable is defined.*
- *If Python gives you an error message on any of the above exercises, try defining x again by entering x=7, and then proceeding.*

Expression	Python's Response	Expression	Python's Response
7a) `>>> x = 7`	no response (Python simply plugs 7 into x)	7e) `>>> x == 7`	`True`
7b) `>>> 4*x`	`28`	7f) `>>> x == 9`	`False`
7c) `>>>x**2`	`49`	7g) `>>>x != 9`	`True`
7d) `>>> x`	`7`	7h) `>>>x != 7`	`False`

Exercise 8
Python also understands inequalities. Enter the expressions below, beginning with 8a), and record the results.

Expression	Python's Response	Expression	Python's Response
8a) `>>>x=3`	no response (Python simply plugs 3 into x)	8c) `>>>x>1 and x>2`	`True`
8b) `>>>not(x<12 and x>2)`	`False`	8d) `>>>x<=3 and not x<2`	`True`

Programs in Python *(suggested time: 30 minutes, excluding optional exercises)*

We can also use the ***Python IDLE Editor*** to write our own programs. To open the IDLE editor from the interpreter, first go to the 'File' menu. Then choose 'New Window'. Due to OS file format differences, some text editors may not display the programs correctly. As such, we suggest you use the Python IDLE editor.

Teacher's Note: *All 12 programs below are packaged into a zip file of Python starter programs. If your students will be running the programs themselves, we recommend providing these files to each student so that they can open them in the* ***Python IDLE Editor*** *on their computers, as opposed to typing them in from scratch.*

Teacher's Note: *When students study and modify computer programs, they are practicing many standards-based problem solving skills. Below are a list of such skills (CA state standards):*

- **Grades 6 and 7, Mathematical Reasoning, Standard 1.1**: Analyze problems by identifying relationships, distinguishing relevant from irrelevant information, identifying missing information, sequencing and prioritizing information, and observing patterns.
- **Grades 6 and 7, Mathematical Reasoning, Standard 1.3**: Determine when and how to break a problem into simpler parts.
- **Grades 6 and 7, Mathematical Reasoning, Standard 2.2:** Apply strategies and results from simpler problems to more complex problems.
- **Grades 6 and 7, Mathematical Reasoning, Standard 3.3:** Develop generalizations of the results obtained and the strategies used and apply them to new problem situations.

Note: If asked to save your program, be sure to save it as a .py file (e.g. ProgramName.py). Once the file is saved as .py, Python will be able to find and run your program. Additionally, Python will know to automatically color code your program for easier readability!

Program 1

Read through and run the program below to improve your understanding of how Python stores values in variables. Notice what happens when the same variable name is used to store more than one value.

Complete the following steps to run the program:

1. Type in the code below and save the file as a .py file (e.g. ProgramName.py)
2. Click on the Run menu at the top of the screen
3. Choose Run Module (or F5 for short)

1a) Predict the answer to the sum of x+y+z below. Then run the following program to test your prediction:

```
x = 4
y = 2
z = 6
```

```
z = 2
print x + y + z
```

Use the table below as a guide to analyze Program 1:

Code	Explanation
`x = 4`	1b) Plug 4 into the variable x
`y = 2`	1c) Plug 2 into the variable y
`z = 6`	1d) Plug 6 into the variable z
`z = 2`	1e) Plug 2 into the variable z, replacing the 6
`print x + y + z`	1f) Add up the values stored in x, y, and z

Teacher's Note: *Analyzing and modifying a computer program provides students with a tangible reason for using variables in math. Point out to students that while Program 2 does not contain a single number, it can compute the product of any two numbers that we enter when the program is run.*

Program 2

Predict and run the program below, and work through the related tasks to improve your programming skills.

Note: Quotation marks and commas are NOT optional in Python. If Python gives you an error message, double check to make sure you included all quotation marks and commas!

2a) Predict what the program below will do. Then run the program to test your prediction:

```
firstnumber = input('enter the first number: ')
secondnumber = input('enter the second number: ')

product = firstnumber*secondnumber

print firstnumber,'x',secondnumber,'=',product
```

Use the table below as a guide to analyze Program 2:

Code	Explanation
`firstnumber = input('enter the first number: ')`	2b) Plug the value that the user enters into the variable called `firstnumber`
`secondnumber = input('enter the second number: ')`	2c) Plug the value that the user enters into the variable called `secondnumber`
`product = firstnumber*secondnumber`	2d) Multiply the value stored in `firstnumber` by the the value stored in `secondnumber`, and plug the answer into the variable `product`
`print firstnumber,'x',secondnumber, '=',product`	2e) Display the value stored in `firstnumber` x the value stored in `secondnumber` = the value stored in `product`

Teacher's Note: *The following tasks are all included in the student version, however completion of these programs is not necessary for students to progress through the rest of this Intro to Python. They serve as extra practice for students to strengthen their understanding of Python syntax and can be assigned as time and student ability allows.*

Task 1: Create new variable names for '`firstnumber`', '`secondnumber`', and '`product`'. Be sure to replace the variables '`firstnumber`', '`secondnumber`', and '`product`' with your own variable names *each time* they appear in the program.

```
newname1= input('enter the first number: ')
newname2 = input('enter the second number: ')

newname3 = newname1*newname2

print newname1,'x',newname2,'=',newname3
```

Task 2: In the print statement at the end of the program, delete the final comma and move the final quotation mark to the end of the line so that it reads as follows: `print firstnumber, 'x',secondnumber,'= product'`

```
firstnumber = input('enter the first number: ')
secondnumber = input('enter the second number: ')

product = firstnumber*secondnumber

print firstnumber, 'x',secondnumber,'= product'
```

Task 3: Change the program to multiply 3 numbers and print the result.

```
firstnumber = input('enter the first number: ')
secondnumber = input('enter the second number: ')
thirdnumber = input('enter the third number: ')

product = firstnumber*secondnumber*thirdnumber

print firstnumber,'x',secondnumber,'x',thirdnumber,'=',product
```

Program 3

Predict and run the program below, and work through the related tasks to improve your programming skills.

Note: On line 3 of the code we divided by 12.0 because if we divide by 12, Python will cut off the decimal part of the answer.

3a) Predict what the program below will do. Then run the program to test your prediction:

```
feet, inches = input('enter your height as follows: feet, inches:  ')

totalinches = feet*12 + inches
totalfeet = feet+inches/12.0

print 'You are',totalinches,'inches tall.'
print 'You are',totalfeet,'feet tall.'
```

Use the table below as a guide to analyze Program 3:

Code	Explanation
`feet, inches = input('enter your height as follows: feet, inches:  ')`	3b) Plug the first value entered into a variable called `feet` and the second value into a variable called `inches`
`totalinches = feet*12 + inches`	3c) Multiply the value stored in `feet` by 12 and add the product to the value stored in `inches`; the sum is stored in a variable called `totalinches`
`totalfeet = feet+inches/12.0`	3d) Divide the value stored in `inches` by 12, add the result to the value stored in `feet`, and store the result in the variable `totalfeet`
`print 'You are',totalinches,'inches tall.'`	3f) Print out the specified height in inches
`print 'You are',totalfeet,'feet tall.'`	3g) Print out the specified height in feet

Teacher's Note: *The following tasks are all included in the student version, however completion of these programs is not necessary for students to progress through the rest of this Intro to Python. They serve as extra practice for students to strengthen their understanding of Python syntax and can be assigned or skipped over as time and student ability allows.*

Task 1: Change the print statements so that they display the results in the following format (as opposed to writing the results out in a sentence), e.g.:

```
feet, inches = input('enter your height as follows: feet, inches:  ')

totalinches = feet*12 + inches
totalfeet = feet+inches/12.0

print 'inches:', inches
print 'feet:', feet
```

Task 2: Add a line of code that will calculate how many centimeters tall the user is, and a print statement that will print out the result (note: there are 2.54 cm in one inch).

```
feet, inches = input('enter your height as follows: feet, inches:  ')

totalinches = feet*12 + inches
totalfeet = feet+inches/12.0
cm = totalinches*2.54

print 'You are',totalinches,'inches tall.'
print 'You are',totalfeet,'feet tall.'
print 'You are',cm,'centimeters tall.'
```

Program 4

Predict and run the program below, and work through the related tasks to improve your programming skills.

4a) Predict what the program below will do. Then run the program to test your prediction:

```
first, second, third = input('enter three numbers, separated by commas: ')

average = (first+second+third)/3.0

print 'average:',average
```

Use the table below as a guide to analyze Program 4:

Code	Explanation
`first, second, third = input('enter three numbers, separated by commas: ')`	4b) Store the first value entered in a variable called `first`, the second value in a variable called `second`, and the third value in a variable called `third`
`average = (first+second+third)/3.0`	4c) Add up the values stored in first, second, and third, and divide the sum by 3
`print 'average:',average`	4d) Print out the average of the three specified values

Teacher's Note: *The following tasks are all included in the student version, however completion of these programs is not necessary for students to progress through the rest of this Intro to Python. They serve as extra practice for students to strengthen their understanding of Python syntax and can be assigned or skipped over as time and student ability allows.*

Task 1: Change the program so that it asks the user to enter the three numbers one line at a time, e.g.:

```
enter the first number:
enter the second number:
enter the third number:
```

```
first = input('enter the first number: ')
second = input('enter the second number: ')
third = input('enter the third number: ')

average = (first+second+third)/3.0

print 'average:',average
```

Task 2: Change the program so that it calculates the average of 5 numbers.

```
first, second, third, fourth, fifth = input('enter five numbers, separated by commas: ')

average = (first+second+third+fourth+fifth)/5.0

print 'average:',average
```

Conditional Logic in Python *(suggested time: 10 - 15 minutes)*

Python uses conditional logic to 'branch out' and handle all possible scenarios to solve a problem. Run the program below three times:

- The first time, enter a number that is less than 2272
- The second time, try a number that is greater than 2272
- The last time, enter 2272

Teacher's Note: *Program 5 below contains the double equals sign ('==') that was introduced in Exercise 8. Remind students that this is not a typo - this is our way of telling Python to check for equality rather than to actually plug a number into a variable. Students can try replacing the '==' sign with a '=' sign, and see for themselves what happens.*

Program 5

Predict and run the program below, and work through the related questions and tasks.

Note: You must include a colon (:) at the end of each if, elif, *and* else *statement, which tells Python to automatically indent the following line.*

5a) Based on the results and the code below, explain what an '`if`, `elif`, `else`' statement appears to do in the space below:

```
guess = input('Guess the average number of texts an American teen sends and
receives each month: ')

if guess == 2272:
   print 'You got it!'
elif guess < 2272:
   print 'Too low; the average American teen sends 2272 texts per month'
else:
   print 'Too high; the average American teen sends 2272 texts per month'
```

Use the table below as a guide to analyze Program 5:

Code	Explanation
`guess = input('Guess the average number of texts an American teen sends and receives each month: ')`	5b) Plug the user's guess into a variable called `guess`
`if guess == 2272:` `    print 'You got it!'`	5c) Python checks to see if the user guessed 2272; if she/he did, Python prints out the words "`You got it!`" If we had used the single equal sign, Python would have pluged 2272 into `guess` and then been confused; the double equal sign tells Python to compare two values, *not* to replace one with the other.
`elif guess < 2272:` `    print 'Too low, try again'`	5d) If the user did not guess 2272, Python checks to see if the value they guessed is less than 2272. If it is, Python prints out the words "`Too low, try again`" (`elif` is short for else if)
`else:` `    print 'Too high, try again'`	5e) If the user's guess was neither correct nor too low, Python prints out the words "`Too high, try again`"

Teacher's Note: *If your students will be copying and pasting program 5f) into the **Python editor** to complete and run, remind them that they must delete all of the underscores (__) before running the program. Python will not run properly until all underscores are deleted and replaced with the correct code.*

5f) In the space below, complete the program so that Python will run as follows:

- Ask the user how many hours she slept last night.
- If she slept 7 - 9 hours, print out 'You are right on schedule'.
- If she slept less than 7 hours, print out 'You need a nap'.
- If she slept more than 9 hours, print 'Time to set the alarm'.

```
hours = input('Enter the number of hours you slept last night: ')

if hours == 7 or hours == 8 or hours == 9:
    print 'You are right on schedule.'
elif hours < 7:
    print 'You could use a nap.'
else:
    print 'Time to set the alarm.'
```

Type your completed program into the ***Python editor*** and run it to check your work.

Programming with Loops in Python *(suggested time: 10 - 15 minutes)*

Python uses a 'while loop' to complete a repetitive task very quickly. A while loop performs a specified task over and over *while* a specified condition is true. Run the two 'blastoff' programs below and describe what appears to be happening in each program.

Program 6: Blastoff outside of the loop	**Program 7**: Blastoff inside of the loop

`number = input('enter a number between 1 and 50: ')`
 	
`while number >= 0:`
`    print number`
`    number = number - 1`
`print 'Blastoff!'`
	`number = input('enter a number between 1 and 50: ')`
	
	`while number >= 0:`
	`    print number`
	`    number = number - 1`
	`    print 'Blastoff!'`

Describe the results of this program: Answers may vary, however students should notice that the program counts down from the number they enter to zero, and then prints "Blastoff!"	How do the results of the program differ now that the print statement is indented? Answers may vary, however students should notice that when the print statement is indented, Python displays the word "Blastoff" after each number in the countdown.

Program 8: We can have Python count the total number of multiples of 3 between zero and one hundred by adding a variable that we chose to call `counter` to our program.

```
x = 3
counter = 0

while x < 100:
    print x
    counter = counter+1
    x = x + 3
print 'There are',counter,'multiples of 3 between zero and one hundred.'
```

Code	Explanation
`x = 3`	8a) Store the value 3 in the variable x
`counter = 0`	8b) Plug the number 0 into the variable called `counter`
`while x < 100:`	8c) Check to see if x is less than 100. If it is, go through the next 3 indented lines. If not, skip all of the indented lines.
`print x`	8d) Print out the value stored in x
`counter = counter+1`	8e) Add 1 to the value stored in `counter`
`x = x + 3`	8f) Add 3 to the value stored in `x`, and replace the previous value with this sum. Then return to the while statement to check to see if the new value stored in `x` is still less than 100.
`print 'There are',counter,'multiples of 3 between zero and one hundred.'`	8g) Display the number of multiples of 3 that are between zero and one hundred

Note:

- *We set `counter` equal to zero at the beginning of the program. We must 'initialize' all variables before using them in a loop. This tells Python that it will need to save some space for the variable before it runs through the loop that follows.*

Once we learn how to program, we can create a personalized calculator that can do our work for us. In the program below, we used a while loop to create a calculator that will solve a word problem:

Teacher's Note: *The following problem is from the 6th grade California CST. The same algorithm used in Programs 9 and 10 applies to many problems involving ratios and proportions. Students who practice this method of listing and counting to solve ratio problems will gain deeper appreciation and understanding of other, more mechanical, ways of solving these types of problems (e.g. cross multiplication).*

Program 9: Mai earns $5.50 per hour at her after-school job. How many hours does she have to work to earn $132?	**Program 10**: Notice that this is the same program, with a print statement is included in the while loop. Compare the results to those of Program 9.

`earnings = 0`
`hours = 0`
`while earnings < 132:`
`    earnings = earnings + 5.5`
`    hours = hours + 1`
`print 'Mai must work',hours,'hours.'`
	`earnings = 0`
	`hours = 0`
	`while earnings < 132:`
	`    earnings = earnings + 5.5`
	`    hours = hours + 1`
	`    print hours,'hours  $',earnings`
	`print '\nMai must work',hours,'hours.'`

Note:

- *In these final programs, `hours` does the same thing that `counter` does in Program 8.*
- *In the final line of Program 10, we also introduced notation `\n`, which tells Python that we would like it to start a new line before printing out the statement that follows.*

Python Outside of Math Class *(suggested time: 10 - 15 minutes)*

Teacher's Note: *The last two programs of this lesson serve as a review of many of the logical and syntactical components of Python. We introduce them as both good exercises in logic and a fun conclusion to the exercises and programs introduced earlier.*

The same programming skills that you learned to use when solving math problems can be used to create games as well.

Program 11

Python has a built-in function called `random` that will pick a random number between any two values that you want. Type the guessing game below into the ***Python IDLE Editor*** and then fill out the table below to help you understand how this program works.

```
import random

guess = input("I'm thinking of a number between 10 an 25.  Guess my number: ")
x = random.randint(10, 25)
while guess != x:
  guess = input('Try again: ')

print 'You got it!  My number is',x
```

`import random`	11a) Tell Python that we will need access to it's random number generator
`guess = input("I'm thinking of a number between 10 an 25.  Guess my number: ")`	11b) Plug the user's guess into a variable called `guess`
`x = random.randint(10, 25)`	11c) Python generates a random number between 10 and 25, and plugs it into the variable `x`
`while guess != x:`	11d) Check to see if the value stored in `guess` does not equal the value stored in `x`
`guess = input("Try again: ")`	11e) Ask for a new value to store in `guess`
`print 'You got it!  My number is',x`	11f) Print out "You got it!, My number is " followed by the random number

Program 12

We can improve our guessing game by providing hints about whether the user's guess is too high or too low. We do this by including conditional logic inside a loop. Fill in the two blank lines of code to tell Python under what condition it should print out '`too low, try again:`'

```
import random
guess = input("I'm thinking of a number between 10 an 25.  Guess my number: ")
x = random.randint(10, 25)
while guess != x:
  if guess < x:
    guess = input('too low, try again: ')
  else:
    guess = input('too high, try again: ')
print 'You got it!  My number is',x
```

Summary

In this lesson we covered several key programming topics that are directly applicable to math curriculum and computational thinking. These include:

- Parentheses affect the order in which Python performs a calculation
- Python stores one value in each variable
- Python uses several symbols rarely found outside of computer science to evaluate mathematical expressions, including
    - ** : exponents
    - % : remainders
    - * : multiplication
    - / : division *(note: expressions must include decimals if the quotient is not an integer)*
    - <= : less than or equal to
    - >= : greater than or equal to
    - != : not equal to
    - == : value comparison
- Python can tell whether an expression is true or false
- We can use the ***Python Editor*** mode to write our own programs
- We can use conditional logic to account for several possible cases of the same problem
- We can use 'while loops' to perform a repetitive calculation quickly and accurately