Python Commands – Input and Output

Command	Description	Examples

`print( )`	Prints onto the screen what is **inside** the brackets.
	
	If what is inside the brackets is enclosed in **quotes**, it will print **exactly** what is inside the quotes.
	
	If a **variable name** is used inside the brackets, it will print the **current value** of the variable.
	
	You can also do **calculations** inside the brackets, and **the result** of the calculation will be printed.
	
	You can also do calculations with **variables,** and again **the result** of the calculation will be printed
	
	You can print several things at once, and they will all come out on the same line. To do this, separate the things you want to print with commas.
		
		`print("Hello world")`
		
		`print(number1)`
		
		`print(5*10)`
		
		`print(number1 + number2)`
		
		`print("Hello ", yourName , " it's nice to meet you")`

`input( )`	Takes input from the keyboard and stores it into a variable.
	
	If **text** is supplied in the brackets **enclosed in quotes**, it will print a message containing exactly what is inside the quotes.
		`getText=input()`
		
		`getName=input("What is your name?")`

`int( )`	If you are expecting the user to type in **a number**, which you want to use later in a calculation, then you must wrap the whole input( ) command inside an **int( )** command	`number1=int(input("Enter a number"))`

`round( )`	Rounds whatever is inside the brackets to the specified number of decimal places. The first thing inside the bracket is the number to be rounded, then comes a comma, and finally the number of decimal places to round to.
	
	Is often used inside `print()` to make the output look neater. Will also work with variables and on calculations inside print.
		`roundedNumber=(round(3.14159,2))`
		
		`averagePerDay=total/7`
		`print(round(averagePerDay,2))`

By H. Mothada. This work is licensed under a Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License.

Python Commands – Selection

Command	Description	Examples

`if … :`	Evaluates a boolean expression to see whether it is **true or false**. You can think of a boolean expression as **like a question** which can only have a true or false answer. When the boolean expression that follows `if` evaluates to true, then the **code** that **follows the colon** (:) will **run**.
	
	If the expression **does not evaluate to true**, all the code between the colon and the end of the **indented block** is **skipped**. In the first example to the right, because the variable `age` contains 14, the expression `age == 14` evaluates to true, and so the two print statements inside the indented block will run. The program then continues on from the first line after the block, in this case the line which prints "End of program". If `age` was any value other than 14, it would skip the two lines in the block and would jump straight to the first line outside the block, which prints "End of program".
	
	The second example is similar but only runs the block after the colon when the `score` variable contains a bigger value than the `highScore` variable. If not, the block is skipped and the code continues at the `print("Game over")` line.
		`age=14`
		
		`if (age == 14):`
		`  print("You are 14")`
		`  print("You must be in high school")`
		
		`print ("End of program")`
		
		`if (score > highScore):`
		`  print("Well done, new high score!")`
		`  highScore=score`
		
		`print("Game over")`

`else:`	`else` is a command that **can** be paired with `if`. It will run the block of code that follows the colon but only if the expression evaluated in the `if` statement evaluates to **false**. Note there is no expression after `else`, just a colon and then an indented block of code that 'belongs' to the `else`.
	
	`else` is **optional**. When you have an `if` you don't need to have an `else`, but when **you have** an `else`, there **must** be an `if` to go along with it.
		`age=14`
		`if (age == 14):`
		`  print("You are 14")`
		`else:`
		`  print("You are not 14")`
		
		`print ("End of program")`

`elif …:`	`elif` is a more advanced version of `else`. Like `else`, it **goes alongside** `if`, and only runs if the expression in the `if` did not evaluate to true. However, unlike `else`, `elif` **does have an expression** that follows it, giving us a chance to ask **another question**.
	
	You can have as many `elifs` as you like, and if *none* of them match, an **optional** `else` clause can be used which runs the indented code after its colon.
		`age=14`
		`if (age == 14):`
		`  print("You are 14")`
		`elif (age==15):`
		`  print("You are 15")`
		`else:`
		`  print("You are neither 14 or 15")`

By H. Mothada. This work is licensed under a Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License.

Python Commands – Iteration (Loops)

Command	Description	Examples

`while … :`	**Repeats** the code that follows the colon for **as long as** the boolean expression evaluates to true. As soon as it evaluates to false, the code inside the block will be skipped, and the program will continue running from the **first line after** the indented block. In the first example the code inside the while block will keep repeating until `i` has the value 10.
	
	In the second example the code will keep asking the user to type something, and will stop when anything other than '`c`' is typed.
		`i=0`
		`while` `(i<10):`
		`  print(i)`
		`  i=i+1`
		`print("End of program")`
		
		`choice=input("Type c to continue")`
		`while` `(choice=="c"):`
		`  choice=input("Type c to continue")`

`range( )`	Returns a set of **integer numbers** that can be **used in a for** loop. If you supply only **one value** in the brackets, it will produce numbers from **zero up to** (but not including) that **value**. If you supply **two** values in brackets, it will produce numbers **from** the first value **up to** (but again **not including**) the second value. If you supply **three** values in brackets, it will produce numbers **from** the first value **up to** (not including) the **second** value, in **steps** of the third value. The step **can be a negative** number if you want to count backwards.	**`range`**`(10)` **`range`**`(1,11)` **`range`**`(10,110,10)` **`range`**`(100,0,-10)`

`for … :`	Repeats the code that follows the colon for as many times as necessary. The variable that follows `for` gets a different value each time around the loop. Often used with a `range` command which returns a set of integers that `for` steps through. In the first example the variable number will get the value 1 first time around the loop, then it will be 2, then 3 and so on until it reaches 11 when it stops and skips to the end of the block.
	
	Another form of the `for` loop will step through each item in a Python list, and the variable that follows `for` will get the value of each item in the list one after the other. In the second example, the value of `item` will be `"pencil"` the first time around the loop, the next time around the value of `item` will be `"rubber"` then next time, `"pen"` and finally `"ruler"`.
		`for` `number in range(1,11):`
		`  print(number)`
		`print("End of program")`
		
		`myList=["pencil","rubber","pen", "ruler"]`
		`for` `item in myList:`
		`  print(item)`
		`print("End of program")`

By H. Mothada. This work is licensed under a Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License.

Python Commands - String Manipulation

Command	Description	Examples

`[ ]`	Gets a single character from a string, after the name of the string, use square brackets to specify which character you would like to extract. Strings are indexed from zero so the first character is at position zero.
	
	In the first example, the character "H" will be put into c as zero has been specified inside the brackets.
	
	In the second example, "o" will be put into c. Specifying a number greater than 4 in this example will cause an error.
		`myString="Hello"`
		`c=myString[0]`
		
		`myString="Hello"`
		`c=myString[4]`

`len( )`	Returns an integer that gives the string length. In the first example `stringLength` will contain 16.	`myString="Computer Science"` `stringLength=len(myString)`

`count ( )`	Count how many times a letter occurs in a string. In the example, `countLs` will contain 2.	`myString="Hello"`
		
		`countLs=myString.count("l")`

`find ( )`	Returns a number giving the position that a letter or word was found in a string (or -1). Can be used with [ ] to extract a character or characters from a string. In the example, `commaPosition` contains 5, because `find` is looking for a comma, and it is found at the 6th character (but we count from zero, hence position 5)	`myString="Smith,John"` `commaPosition=myString.find(",")`
`[ : ]`	Gets multiple characters from a string, after the name of the string, use square brackets to specify which part of the string you would like to extract. The first value before the colon is the start position and the second value after the colon specifies how many characters should be extracted from that position. In the example, the characters "He" will be put into `subString` characters from 0 up to (but not including) 2 have been extracted.	`myString="Hello"` `subString=myString[0:2]`

By H. Mothada. This work is licensed under a Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License.

Python Operators

Below are some symbols you will commonly need to use in Python programs to perform various operations

Symbol	Purpose	Example
=	Store something in a variable	`myAge=13`
==	Compare two values to each other	`if (myAge == 16):`
+	Add two numbers	`print(number1+number2)`
-	Subtract two numbers	`print(number1-number2)`
*	Multiply two numbers	`print(number1*number2)`
/	Divide two numbers	`print(number1/number2)`
( )	Do the calculation inside inner brackets first	`print ( (mark/totalMarks) *100 )`
#	To allow you to write notes (comments) in your program	`# This program was tested on 02/05/2018`

Common errors

If your program is not working as you expect or you get an error message check the following:

Check	Example	Error in example
Have you closed the quotes in a print or input command?	`print("Hello world)`	Missing `"` after `world`
Did you put a single equals when you meant double?	`if(age=13):` `  print("You are 13")`	Missing = sign. `age==13` compares `age` to 13, `age=13` *sets* it to 13
Do you have an equal number of open and closed brackets?	`age=int( input("Your age?")`	Missing closing bracket at the end of the line
Have you forgotten a comma where you need one?	`print("Hello " name)`	Missing comma between `"Hello "` and `name`
Have you forgotten to put int( ) around your input statement if you are expecting a number to be input?	`age=input("Your age?")` `age=age+1`	Missing `int()` around the input statement
Have you misspelled a variable name?	`name=input("Your name?")` `print("Hello ",naame)`	Variable has been misspelled on the second line as `name`
Have you indented blocks of code correctly?	`if a>b :` `print ("a is bigger than b!")`	Python uses spaces to decide whether code belongs to a block within an `if`, `else`, `elif`, `while` or `for`. The print line should have spaces to ident it inside the `if` block

By H. Mothada. This work is licensed under a Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License.