ï»¿

Python Concepts for the Beginners

The Smarty Party 11366

1. Installing and Running

To start writing Python code you are going to need an editor and an interpreter. An editor is where you write your code, the interpreter is what runs it. There is an online editor and interpreter is repl.it. You can write code in repl.it and then it shows the results in the browser. This is an extremely accessible option for many people since it can be used anywhere there is an internet connection and a modern browser.

If you would like to play with the language on your computer locally you need to download and install the language at https://www.python.org/downloads. Once the language is installed you can either use notepad to write the python code or you could use one of the IDEâs recommended at https://wiki.python.org/moin/IntegratedDevelopmentEnvironments.

2. Basic Programming Concepts

Variables

Variables are the things that are used as an alias for a value. It allows you to treat numbers and other things as a concept that you can change and manipulate in different parts of the program as opposed to a raw value. It is very useful and necessary even in some of different most basic programs you can make. A few examples are below:

```
variable_name = 7
#OR
variable_name_again = "this is a string"
```

There are two data types you will be using as a beginner: strings and integers. Integers are whole numbers, you don’t need any quotations or brackets for them to register, they stand on their own. Strings are a string of characters it can be a sentence or a bunch of numbers it doesn’t matter as long as it’s within quotation marks like this: "hey good job, you got this far".

Booleans

Booleans are simple true false statements that help people control the flow of a program. Some operators that are used help determine if an integer or input are equal to, greater than or less than a value. These are often used in loops and in If-Else statements but we will get to these later in this document.

< <-- this is the greater than operator. It is used to say "if A is larger than B" or "A < B"

> <-- this s the lesser than operator this i used to sat "if A ls smaller than B" or "A > B"

! <-- this is the not operator it says â A is not Bâ or within context âA ! Bâ

== <-- this is the equal to operator it is used to say "if A is equal to B" or "A == B"

<= <-- this is the equal to or greater than operator is is used to say "if A is equal to or greater than B" or "A <= B"

>= <-- this is the lesser than or equal to operator it is used to say "if A is less than or equal to B" or "A >= B"

!= <-- this is the not equal operator it is used to say "as long as A and B are not the same" or within context "A != B"

P.S. The reason we use == instead of one equal sign for the equals to operator is because the single equal sign is used to assign value not to check it.

Math and Stuff

Coding is just a bunch of math and breaking processes down into a very specific set of instructions so naturally you would have functions that perform basic arithmetic.

+ <-- this is the addition operator it adds two integers together for example: "A + B"

- <-- this is the subtraction operator it subtracts two integers for example: "A - B"

/ <-- this is the division operator it divides two integers for example: "A / B"

* <-- this is the multiplication operator it multiplies two integers for example: "A * B"

% <-- this is the modulo operator it divides two integers and returns the remainder for example: "10 % 3" --> 1

()<-- these are parenthesis they specify the order the computer will solve the math problem for example: "(9/3) + 7" --> 10

Strings

Strings are not integers they are well, strings of characters that have no numerical value. For a set of characters to recognised as a string they need to be surrounded by quotation marks like so: "Hello". To be able to use the operators below on a string it helps to have assigned the string to a variable or alias like so: alias = "Hello".

len() <-- this is the length operator it gets the length of whatever is within the parenthesis

str() <-- this makes the item within the parenthesis a string while its within the parenthesis

lower() <-- this makes every letter in the string lowercase it can also be used like this alias.lower()

upper() <-- this makes every letter in the string uppercase it can also be used like this alias.upper()

Loops

Loops are used to repeat commands so long as the conditions of the loop are met. This is useful for knocking out tasks quickly and it also makes code easier to read. Loops either operate by checking criteria you specify to the right of the loop type. The criteria is either a range of times that the loop will be repeated or a boolean statement that can be evaluated to true or false and will run of the statement specified evaluates

to true or false.

For-Loops

For loops run for a determined amount of time. In this particular case the for loop will run four times. The range specified will determine how many times it will be run. You could use a variable for either end of the range. For example:

```
for x in range(0, 3):
        print("pretend these are answers to the math test")

#OR

B = 1
user_input = str(raw_input("Type a word in here or something: "))
for x in range(0, len(user_input)):  #this loop will go on from 0 to the length of
        print("we are on character " + B + " of the thing you typed in0)
        B = B + 1 # you could use use B += 1 for the same effect
```

While-Loops

While Loops are just for loops but it does something while the specified criteria is true or false. The while loop checks a condition if it meets the criteria defined it will loop working through the loop and move on down the code but if it doesn't meet the criteria it will continue to loop until it does. An example could be written like:

```
while 'apples' != 'oranges':
        print("don't compare apples and oranges")
```

Break

Breaks "break" out of a loop or flow of events if something isn't going as planned or if an event specified occurs. This is used when troubleshooting, for safety reasons when programming machine that moves, or in other situations where you decide that switching loops might be helpful. It is considered good practice to use these when utilising a loop, for example:

```
while True:
        input = raw_input(âHey Dave, what is your name: ")
        if input.strip != 'Dave':
                break
```

If & Else Statements

If and Else statements are going to be your best friend for a while. They ask if a situation meets criteria A if it does the if line runs if it doesn't it moves on down the line. Think of it as yes or no questions in a flow chart it controls in what direction a program flows based on the situation at hand.

if <-- this statement is necessary for any of the other "if and else" statements to exist. It tests if a condition is true or false if itâs true it executes the code within parenthesis, for example:

```
if (2 != 1){
        print("two isn't one")
```

```
}
```

elif <-- This is what you sandwich between the Else and If statements. This needs If and Else to exist and you can have as many of these as you want. You do need to specify conditions though just like the "If" statement. It goes in between the if and else statements as the name would suggest. For example:

```
elif (2 == 2) {
        print("two is two")
}
```

else <-- This is the else operator and it is the final operator you could use for if and else statements. It always comes last and it doesn't need any criteria specified beside it because it is the operator that runs if no other criteria that was specified worked. An example of an else statement is below:

```
else {
        (print "this is the last possible outcome")
}
```

Arrays / Lists

An array is a list of characters attached to numeric values. The start from 0 and move up according to how many items you place in the array. You create an array or list by putting items within brackets separated by commas. For example:

```
Names_for_array = ["List","Array","things in brackets","references"]
```

You can retrieve items from the array by naming the array in this case âNames_for_arrayâ with brackets around the number of the item you would like to retrieve. For example if I wanted to print array I would type:

```
print(Names_for_array[1])
```

Comments

Comments are things that the interpreter will ignore. These are put here to help yourself and other people be able to read your code without running it. Think of it as little notes attached to the code to clarify the function of certain parts

```
# <-- use at the beginning of a line for a single line comment

      OR

""" <-- use at the beginning and ending of a multi-line comment (three quotation ma
```

3. Your First Program

```
        print("HELLO FRIEND, HOW ARE YOU")
```

About Your First Program

The print command prints the sting you put within the parenthesis. The string you put in the parenthesis is called an "Argument" it kinda specifies what if going on to the function in this case the function was print(). This is one line and it is the beginning for your possible time with python. Now that you have made a hello world in python feel free to put it on your resume you have gotten a basic grasp of and idea of what a computer language is like in theory.

4. Extra Concepts

Importing Libraries

To import a library first you should make sure it exists. The python standard library and the documentation has a list of libraries the exist and their functions. After deciding what package you want use and reading about how to use it you need to import the library at the beginning of your code. For example say you want to import the os library, the import command would look like this:

```
import os
os.system("echo I’m speaking though the terminal")
```

Dictionaries

Dictionaries let you substitute certain values or characters for new ones. It checks a stream or set of characters and if properly directed replaces the characters found with a valid replacement. A dictionary would look like this.

```
Dictionary={’a’:’A’,’b’:’8’,’c’:’&’,’milk’:’supreme’} #etc.
```

Getting Input

Getting input can be accomplished by using the "raw_input" function. After you get the raw input I would recommend attaching this input to a variable to make it easier to work with. For example:

```
user_tea_input = raw_input("Hello chaps, what is your favorite tea?")
```

Random Number Generation

Random number generation could be helpful while making a math test. You use the randint operator from the random library to randomly generated numbers within a range then you assign that randomly generated value to a variable. you can substitute that variable in a place where a number would be to create a randomly generating integer. For example:

```
import random
random_value = randint(A, B)
print("randy has " + random_value + " pieces of candy")
```

String Interpolation

Strings are strings and variables are variables but sometimes got gotta merge the two and thatâs what you call string interpolation. It’s just slipping a variable into a string. This can be useful when changing

values in a string in accordance to user input. For example:

```
input0=str(raw_input(âput in your name: â))
input1=raw_input(âhow old are you: â)
print(âyour name is %s and you are %d years oldâ % (input0, input1))
```

The reason the letter after the percent sign is there is because it correlates to the data type itâs displaying, s for string and d for digit. The order of the variables placed out of the sting matters because that's the order the signs will be replaced. If you mix them up then you might end up with variables where they shouldn't be.

5. Some Ideas For Practice

- Math test that takes input and tells you if you were right or wrong
- A High low game using loops
- calculate the Fibonacci sequence and print each number on a new line
- make a tic-tac toe game
- text based Choose your own adventure