

Oleg Gleizer
bungo by bungo bungo

1 Intro to Python

1.1 Variables

A variable is a container that stores some data value. In the code below, we create two variables, `x` and `y`, which stores the data values of 100 and "Math Circle", respectively. We can also use the `print` function to output whatever those variables are storing. Try it out yourself!

```
1 x = 100
2 y = "Math Circle"
3
4 print(x)
5 print(y)
```

[Note: to give variables a value involving characters, which we call a “string”, you need to surround the value with quotes - otherwise you’ll get an error - but those quotes aren’t actually considered part of the value]

However, be careful with `print` statements - if you want to print a variable’s value, you simply enter the variable name in the parentheses. Meanwhile, if you want to print a specific word or character you surround the value you want to print with quotes.

```
1 x = 100
2 print(x) -> 100
3 print("x") -> x
```

The value stored by a variable is not fixed and can be updated or changed to something else. (this is what we’d call “overwriting”)

Problem 1 *Observe the code below. Can you predict what will be printed?*

```
1 x = 9
2 x = "AAAAAHHHHH"
3
4 print(x)
```

Variable names are case sensitive. For example, `var` will not be overwritten in the following example.

```
1 var = "Hello"
2 VAR = "World"
3
4 print(var)
```

1.2 Lists

Variables are only capable of holding one data value at a time. But what if we wanted to store more than one? Lists allow us to do just so. Note the syntax of the example below. We separate the different data values by using commas while also surrounding it all with brackets.

```
1 listofthings = [5, 10, "banana", 10]
```

Lists can contain a mix of both numbers, strings, and any other data type. They are also ordered, and you can call on a specific data value within a list based on its position, known as the *index*. Try out the following code!

```
1 groceries = ["banana", "melon", "milk", "bacon", "eggs"]
2
3 print(groceries[0])
4 print(groceries[1])
5 print(groceries[2])
6 print(groceries[3])
7 print(groceries[3])
```

Note how when we want to call the first data value `"banana"`, we use an index of 0 rather than 1. This is because Python is a zero-indexed language. That just means that Python starts counting up at 0. So the first position correlates to an index of 0, the second position with an index of 1, third position with index 2, and so on.

1.3 Math and Operators

Python also has a bunch of built in operators to run mathematical calculations with. For example, here is a table of some operators.

Name	Operator	Example	Output
Addition	+	3 + 2	5
Subtraction	-	5 - 3	2
Multiplication	*	4 * 2	8
Division	/	8 / 2	4
Exponentiation	**	2 ** 3	8

Problem 2 *Sometimes, updating variables can look very different from what we're used to in math. What will be printed at the end of this code? (The formatting for the print statement will print the value of a, then a space, then the value of c.)*

```
1 a = 2
2 b = 5
3 c = 25
4 a = a + b
5 c /= 5
6 print(a, " ", c)
```

Problem 3 *Using the operators above, solve the following quadratic equation. Recall the quadratic formula!*

$$3.84x^2 + 8.26x - 11.76 = 0$$

1.4 Conditional Statements

A conditional statement is a way for your code to make decisions based on if some condition is true. Observe the following example:

```
1 if 5 > 3:
2     print("5 is greater than 3")
```

Here, the `if` statement will run and print the code inside of it as the condition, `5 > 3`, is in fact true. If it were not true, then anything inside the `if` statement would not run at all.

Besides from `>` and `<`, there are other logical conditions we can try out:

Name	Operator
Equal to	<code>==</code>
Not equal to	<code>!=</code>
Greater than or equal to	<code>>=</code>
Less than or equal to	<code><=</code>

In Python, an `if` statement can also have an `else` clause. The `else` statement will only run when the `if` statement above it is false.

```
1 if 3 > 5:
2     print("3 is greater than 5")
3 else:
4     print("since the above if statement is false, I will be printed")
```

An `if-else` statement allows us to write code that can choose between two choices. But let's say we wanted to make a choice between more than two? Well, we can do so by adding in the `elif` keyword.

```
1 number = 7
2
3 if number == 5:
4     print("The number is 5")
5 elif number == 7:
6     print("the number is 7")
7 else:
8     print("the number is not 5 or 7")
```

Let's walk through what is happening in the code:

- `if number == 5:` - this checks if the variable, `number`, is 5. If so, the code will execute the body. After doing so, it skips the next `elif` and `else` statement.
- `elif number == 7:` - if the first condition above is not true, Python will then check this one. If it is true, it executes the the code in it, promptly skipping the `else` statement after it.
- `else:` - if the first two conditions are not true, then Python will default to this `else` statement and execute whatever is in it.

1.5 Loops

A loop is a way to repeat a block of code multiple times. It helps you do the same task over and over again without writing it again and again. There are two main types of loops. A for loop, and a while loop. Let's go over the former.

A for loop iterates over the same block of code based on a certain condition.

The most basic version repeats a line of code while iterating over a list. In other words, it goes through each value in a list, one-by-one and one at a time. A for loop always follow the format of `for _ in _:`. Let's see what this means:

```
1 groceries = ["banana", "melon", "milk", "bacon", "eggs"]
2
3 for grocery_item in groceries:
4     print(grocery_item)
```

Here, we have a list called `groceries`. When we write a for loop, we also have to define a iteration variable, in this case `grocery_item`. During the first iteration, the variable `grocery_item` is set to the first value in the list, `"banana"`. Then the code within it runs, printing out `"banana."` Then the for loop restarts, with `grocery_item` next being set to the second value, `"melon"`, with the code inside of it running again and printing `"melon."` This continues until there are no more values in the list.

We can do the same, this time printing out sequential numbers as well:

```
1 numberlist = [0, 1, 2, 3, 4, 5]
2
3 for number in numberlist:
4     print(number)
```

We can achieve the same result as above by using the `range()` function as `range(6)` automatically creates a list of numbers from 0 to 5.

```
1 for number in range(6):
2     print(number)
```

The `range()` function also allows us to determine how many times we want to loop some piece of code just by inputting the desired number.

```
1 for i in range(10):
2     print("I'm being printed 10 times!")
```

1.6 Functions

A function is a strategy to avoid copying code over and over again, which both takes up space and makes things harder to read. The format of a function follows the format of `def functionname():`, where `functionname` can be whatever name you want it to be.

```
1 def bungo():
2     print("bungo")
```

Above, we created a function. But it won't execute the code inside it. This is because all we've done is just define it. In order to execute the function, we need to call it as well:

```
1 def bungo():
2     print("bungo")
3
4 bungo()
```

A notable thing about functions is that it can take in inputs, or arguments.

```
1 def multiplybytwo(inputnumber):
2     twotimes = inputnumber * 2
3     print(twotimes)
```

```
4
5 multiplybytwo(5)
```

In the above example, we defined a function called `multiplybytwo` with the argument `inputnumber`. When we call the function, we have to also include what our argument for the function is, in this case 5.

Now, every time we want to multiply a number by two, we can call the function instead of entering the formula. This might not seem like much of an advantage here, but imagine a 20 line function - by calling it instead of just reusing the code, you're saving so much space!

In functions, we can add in as many arguments as we want. Also, if we want to output the result of our function without printing, we can use the `return` keyword. In the below example, `var2` will be the output of our function, and we save it to the variable `somevariable`.

```
1 def domath(a, b, c):
2     var1 = a + b
3     var2 = var1 / c
4     return var2
5
6 somevariable = domath(5, 7, 8)
```

2 The Turtle Module

Sometimes the commands in basic Python aren't enough. To make things easier, different people have created their own collections of commands, called "libraries", that anyone can use. A common one is `numpy`, which has a built-in square root that you can use. In our case, we will be using the `turtle` library.

Problem 4 *Type in the following commands. Hit ENTER after entering each command. See what happens.*

```
1 from turtle import *
2 fd(100)
3 rt(90)
4 fd(200)
```

You can clear the above picture using the following prompt.

```
1 clear()
```

Note that the `clear()` command clears the picture, but does not revert the turtle to the original position. The `reset()` command does just that.

To draw a square, do the following.

```
1 for i in range(4):
2     fd(100)
3     rt(90)
```

Then run your code.

Problem 5 *In the Turtle module, draw an equilateral triangle.*

Problem 6 *In the Turtle module, draw a beautiful picture of your own.*

We can write a function to draw a square like so:

```
1 def square():
2     for i in range(4):
3         fd(100)
4         rt(90)
```

Problem 7 *Type the following few lines of code. What do you think is going to happen when you run the program?*

```
1 for i in range (36):
2     square()
3     rt(10)
```

Problem 8 *What would we do if we want the turtle to draw a square with a side length different from 100 units?*

Problem 9 *Change the `square()` function so that it draws a square based on a given side length.*

Problem 10 *Run the following prompts. This is to check if your square function works.*

```
1 square(80)
2 square(100)
3 square(120)
```

Problem 11 *Type in the following lines of code. What happens?*

```
1 side=20
2 for i in range(30):
3     square(side)
4     rt(5)
5     side=side+10
```

Problem 12 *Let n be the number of sides of a regular n -gon with a side length s . Define `polygon(s,n)` as a function of the variables s and n . Use the function to draw a regular pentagon and hexagon*

In the Turtle mode, there exists a command `circle(r)` that draws a circle of radius r .

Problem 13 *Assume that the command `circle(r)` does not exist. Define a function `circle(r)` that draws a circle of radius r yourself.*

Problem 14 *Do you think the following Python code would draw a circle of radius r ? Why or why not? Try to answer the question without running the code.*

```
1 def circle (r=80):
2   for i in range(1000):
3     fd(3.1416*r/500)
4     rt(.36)
```

Problem 15 *Change the code of the function `circle(r)` to increase the speed of drawing without lowering the visual quality of the picture.*

Problem 16 *Using the Turtle module, write the code for the function `petal(r)` that draws two quarter-circles of radius r forming a petal.*

In programming, a function used by another function is called a *subroutine*.

Problem 17 *Using the Turtle module, write a function `flowerhead()` that uses the subroutine `petal(r)` to draw a flowerhead of twelve congruent petals equally spaced around the common center.*

We have learned the following four major tools of programming.

LOOPS

FUNCTIONS

VARIABLES

SUBROUTINES

Problem 18 *In your own words, explain what a programming loop is and what it is good for.*

Problem 19 *In your own words, explain what a function is in programming and what it is good for.*

Problem 20 *In your own words, explain what a variable is in programming and what it is good for.*

Problem 21 *In your own words, explain what a subroutine is in programming and what it is good for.*

3 Input, output, and the *while* loop

In the following example, we will make the computer learn your name (or any other). Type in the following line of code.

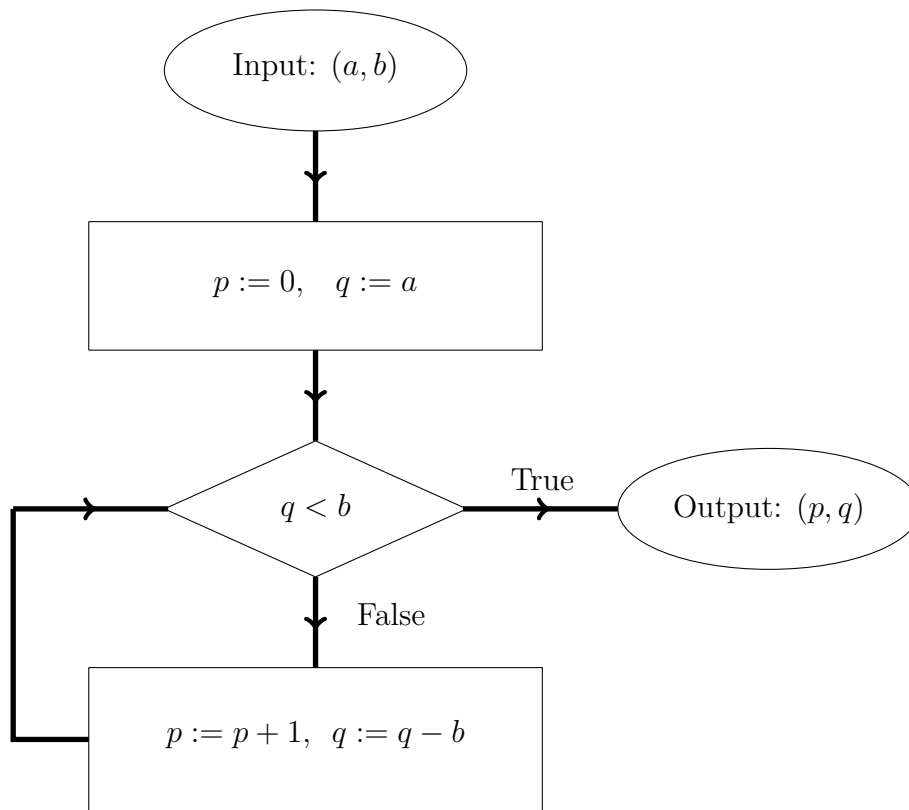
```
1 name = input("What is your name? ")
```

It assigns whatever you type in the input to the variable `name`.

Problem 22 *Use the command `print` and the variable `name` to make the computer greet you after you tell it your name.*

We will see more input and output options later. Below, we will also need the *while* loop construction that is quite self-explanatory.

The following is the algorithm that takes two positive integers, a and b as inputs and finds the positive integers p and q such that $a = pb + q$. In other words, it divides a by b and finds the quotient p and the remainder q , $a \div b = p \text{ rem } q$.



Note that division is implemented as repeated subtraction!

Problem 23 Please apply the algorithm, pen on paper, to the numbers $a = 7$ and $b = 3$.

The following Python code seems to be a perfect implementation of the algorithm.

```
1 a = input("Please enter a positive integer a you'd like to divide. ")
2
3 b = input("Please enter a positive integer b you'd like to divide a by. ")
4
5 p = 0
6 q = a
7
8 while q >= b:
9     p = p+1
10    q = q-b
11
12 print("p=", p, " q=", q)
```

Problem 24 *Try to run the code in Python. What's wrong? Can you correct the code?*