**Intro to Python: Cheat Sheet**

If you get stuck while coding in Python, feel free to use this document to find some of the functions and other things that we learned about.

**Strings** – Page 2

**Integers** – Page 3

**Variables** – Page 4

**Inputs** – Page 6

**Booleans** – Page 7

**Lists** – Page 10

**Loops** – Page 12

**Functions** – Page 14

**Strings**

What are strings? In Python, strings are just a fancy way of describing data that is *text*. You can tell that something is a string when it is wrapped in “quotation marks”. Some examples of strings are:

“Hello, World!”

“I love Code Ninjas”

“123abc”

With a string, you can use the print function. The print function allows us to print text onto our screen.

*Example*:

>> print(“Hello, World!”)

Hello, World!

You can connect two strings together by putting a plus sign (+) between the two strings. When you’re connecting two strings, make sure that you include a space in one of your strings! If you don’t, the words in your strings will be stuck together!

*Examples:*

>> “Hello, ” + “World!”

‘Hello, World!’

>> “no” + “space”

‘nospace’

You can use a print statement to print two strings that are connected together.

*Example*:

>> print(“Super ” + “Mario”)

Super Mario

**Integers**

What are integers? In Python, integers are just whole numbers, like 1, 10, -25, or 123456. With these integers, Python can do basic math, like adding (+), subtracting (-), multiplying (\*), and dividing (/).

*Examples:*

>> 3 + 4

7

>> 0 – 5

-5

>> 2 \* 2

4  
>> 12 / 3

4.0

You can find the largest number in a set of numbers by using the max function, or the smallest number in a set of numbers by using the min function.

*Examples:*

>> max(9, 12, 0, 2)

12

>> min(12, 7, 11, 19, 10, 8, 4)

4

You can print an integer using the print function. We can also connect a string and an integer together, but first, we need to convert the integer into a string. To do this, we need to use the str function. If we forget to use the str function, we will get an error!

*Examples:*

>> print(10)

10

>> print(“McDavid wears number ” + str(97))

McDavid wears number 97

**Variables**

Variables are used to store data and give them a name so that you can use them later in your code. To create a variable and **assign** a value to it, you need to use the equal sign (=) and the following format:

variable = value

*Examples:*

>> x = 10

>> x

10

>> name = “Tristin”

>> name

‘Tristin’

There are many ways to use variables, but the three most common ways to use them are to **perform a computation**, **compute a new variable**, and to **update variables**.

*Examples:*

>> num = 11

>> print(num + 16)

27

>> new\_num = num + 12

>> new\_num

23

>> num = num + 10

>> num

21

>>

>> message = “of”

>> print(message + “ten”)

often

>> new\_message = message + “fer”

>> new\_message

offer

>> message = message + “ course”

>> message

‘of course’

**Inputs**

To get input from the player that you can use in your program, you can use the input function. Inside of the input function, type the question or prompt that you want to give the user inside of a string, and this question or prompt will be given to the user to answer when they run the program. Remember to include a space at the end of your prompt!

As well, to use this user input somewhere else in your program, remember to store it inside of a variable!

*Example*:

>> name = input(“What is your name? ”)

What is your name? Tristin

>> print(“Hello, ” + name + “!”)

Hello, Tristin!

You can also ask the user for integer inputs, but you need to convert the input from a string to an integer first. To do this, you can use the int function.

*Example*:

>> year = input(“What year were you born? ”)

What year were you born? 2002

>> year = int(year)

>> age = 2025 – year

>> print(“I think you’re ” + age + “ years old.”)

I think you’re 23 years old.

**Booleans**

In Python, Booleans are used to tell us if a condition or variable is either True or False. Remember that Booleans can only be True or False, there’s nothing in between!

We can compare numbers using greater than (>), less than (<), equal to (==), not equal to (!=), less than or equal to (<=), and greater than or equal to (>=). Checking these comparisons will give us a Boolean result.

*Examples*:

>> 8 > 3

True

>> 7 < 4

False

>> 10 == 10

True

>> 7 != 6

True

>> 6 <= 8

False

>> 3 >= 3

True

>> 5 >= 3

True

You can also make more complex Boolean logic statements by using the Boolean connectors and, or, and not.

* and – is True only if both comparison statements are true
* or – is True only if at least one comparison statement is true
* not – changes True to False, or changes False to True

*Examples*:

>> x = 15

>> x > 10 and x < 20

True

>> x > 0 and x < 10

False

>> x > 10 or x < 0

True

>> x > 20 or x < 10

False

>> not True

False

Usually, Booleans are used to decide whether certain things should happen in your code or not. These Booleans can be used as conditions in if-elif-else statements. An if-elif-else block can be built like this:

A computer screen shot of a code

AI-generated content may be incorrect.

*Example*:

>> lives = 2

>> score = 73

>>

>> if lives == 0:

>> print(“You lose!”)

>> elif score > 100:

>> print(“You’re an expert!”)

>> elif score > 50:

>> print(“You’re pretty good.”)

>> else

>> print(“You need practice...”)

You’re pretty good.

**Lists**

In Python, lists are used to store a collection of values together in one container. You can tell that you have a list of things when they are wrapped in [square brackets]. The values that we store inside of a list are called **elements**. Elements of a list are separated by commas. A list can be made with many different types of elements.

*Examples:*

>> num\_list = [1, 4, 9, 16]

>> str\_list = [“Hello”, “World”]

>> mix\_list = [“RNH”, 93, True]

Lists also have a few common functions that we can use:

* len – returns the length of the list
* append(item) – adds the given item to the list
* pop() – removes and returns the last item in the list

*Examples*:

>> subjects = [“la”, “math”, “science”]

>> len(subjects)

3

>> subjects.append(“computers”)

>> subjects

[“la”, “math”, “science”, “computers”]

>> len(subjects)

4

>> last = subjects.pop()

>> print(“We just removed ” + last)

We just removed computers

>> subjects

[“la”, “math”, “science”]

You can also get elements from a specific location in a list by **indexing** the list. To index a list, you use the list’s name, square brackets, and the index (position) of the element that you want to get from the list:

listname[index]

Remember that lists start counting at 0! So, the first element of the list has index 0, the second element of the list has index 1, the third element has index 2, and so on.

*Examples*:

>> rooms = [“kitchen”, “living room”, “bathroom”, “bedroom”, “pantry”]

>> rooms[0]

‘kitchen’

>> rooms[3]

‘bedroom’

>> rooms[1]

‘living room’

**Loops**

Loops can be used to execute the same section of code many times in row. There are two different kinds of loops that we can use in Python: the for loop and the while loop.

A for loop should be used when we know how many times we want to repeat our code. There are two types of for loops:

1. for variable in range:

* This will execute the loop over a **range** of numbers, one time for each number

1. for item in list:

* This lets us do something with every item inside a list, executing the loop one time for each item in the list

*Examples*:

>> for num in range(1, 4):

>> print(num \* num)

1

4

9

>> numbers = [1, 5, 2, 8, 10]

>> sum = 0

>> for number in numbers:

>> sum = sum + number

>> print(sum)

26

Remember that there are three types of range:

1. range(stop) – a list of all numbers between 0 and, up to but not including, the stopping number
2. range(start, stop) – a list of all numbers between the starting number and, up to but not including, the stopping number
3. range(start, stop, step) – a list of numbers between the starting number, increasing by the stepping number each time, and up to but not including, the stopping number

*Examples:*

>> list(range(5))

[0, 1, 2, 3, 4]

>> list(range(3, 6))

[3, 4, 5]

>> list(range(1, 12, 2))

[1, 3, 5, 7, 9, 11]

A while loop should be used when we *don’t* know how many times we want to repeat our code. This loop will continue to execute until the Boolean condition that we define with the loop becomes False. Make sure that this condition can eventually become False! Otherwise, we’ll be stuck in the loop forever!

*Example*:

>> turns = 3

>> while turns > 0:

>> print(“You have ” + str(turns) + “ left.”)

>> turns = turns – 1

You have 3 turns left.

You have 2 turns left.

You have 1 turn left.

**Functions**

Functions are sections of code that we give a name to so that we can use it anywhere in our program! To create a function, we need to use a function definition.

def function\_name():

After we define a function, we can use the function’s code anywhere else in our program by **calling** it.

function\_name()

*Example*:

>> lives = 3

>> score = 500

>> def hit\_enemy():

>> lives = lives – 1

>> score = score – 50

>> print(“Lives: ” + str(lives))

Lives: 3

>> print(“Score: ” + str(score))

Score: 500

>> hit\_enemy()

>> print(“Lives: ” + str(lives))

Lives: 2

>> print(“Score: ” + str(score))

Score: 450

You can also give your function information to use by giving it **parameters**. Parameters are special variables that you add between the brackets of the function definition and can only be used inside of the function that you define them in. A function can have as many parameters as you want, just remember to separate them with commas.

def function\_name(param1, param2)

To assign values to these parameters and give the information to your function, you need to use **arguments** in function calls. Arguments are the information that you want to your function to use, and you should put them between the brackets of your function call.

function\_name(arg1, arg2)

You can also use functions to compute values, then give the result back to your code to use by using a return statement. A return statement will immediately end your function and give your code a result that you can use somewhere else! Remember to save it to a variable!

*Example:*

>> def rect\_area(width, height):

>> area = width \* height

>> return area

>> w = 4

>> h = 5

>> a = rect\_area(w, h)

>> print(“A rectangle of width ” + str(w) + “ and height ” + str(h) + “ has area” + str(a) + “!”)

A rectangle of width 4 and height 5 has area 20!