**Python**

**Print**

In Python the print() function is used to tell a computer to talk;

print("There is something at work in my soul, which I do not understand.")

The printed words are referred to as **output**.

Here is another example of a simple command;

print("Hello world!")

**Strings**

Blocks of texts are strings. For example, the previous code blocks are examples of strings.

**Variables**

Programming languages offer a method of storing data for reuse. Perhaps we sant to store a greeting, we can do this in a variable.

message\_string = "Hello there"

# Prints "Hello there"

print(message\_string)

In this instance, we stored a string as a variable. When we run our code, python will print the message\_string variable which is; ‘Hello there’.

* Variables do not have spaces or symbols but they\_can have underscores.
* Variables should not begin with numbers, but they can\_have\_numbers\_after\_the\_first\_letter\_23432432.

Variables can be updated.

# Greeting

message\_string = "Hello there"

print(message\_string)

# Farewell

message\_string = "Hasta la vista"

print(message\_string)

Imagine that Python is working chronologically in a descending order. Initially there first message that will be printed will be ‘Hello there’.

Later, we re-assign the variable; message\_string to say ‘Hasta la vista.’ This will then print ‘Hasta la vista’ henceforth until the variable is updated.

**Errors**

^

Python will highlight errors with this symbol. We call the process of updating the program so that it no longer produces unexpected errors ‘debugging’.

There are two common errors in Python; SyntaxError and NameError.

Syntax Error

This means something is wrong with the way in which the program is written – punctuation that does not belong, a command where it is not expected, or a missing parenthesis can all trigger a SyntaxError.

NameError

A NameError occurs when the Python interpreter sees a word it does not recognise. Code that contains something that looks like a variable but was never defined will throw a NameError.

print('This message has mismatched quote marks!")

print(Abracadabra)

Here are two examples. Firstly we have a SyntaxError. This is because the quotation marks do not match.

Secondly we have a NameError. This is because we do not have a variable assigned to Abracadabra. If we are trying to print Abracadabra on its own, then we need quotation marks!

This code will work:

print('This message has mismatched quote marks!')

print('Abracadabra')

**Numbers**

Computers can also use numbers. There are multiple ways to store numbers.

An integer or int is a whole number. There are no decimal points.

A floating-point number, or a float, is a decimal number. It is used to represent fractional quantities as well as precise measurements.

Numbers can be assigned to variables.

an\_int = 2

a\_float = 2.1

print(an\_int + 3)

# Output: 5

In this instance when we print; the number 3 is a literal. This means it is not a variable, it is ‘literally’ the number 3.

**Calculations**

Python performs arithmetic operations with; +, -, \* and /.

Here are some arithmetic operations:

# Prints "500"

print(573 - 74 + 1)

# Prints "50"

print(25 \* 2)

# Prints "2.0"

print(10 / 5)

When we perform division python converts all int numbers to float s.

**Changing Numbers**

Variables that are assigned numeric values can be treated as if they are numbers.

Performing arithmetic operations on a variable does not change the variable. You can only update a variable using the = sign.

coffee\_price = 1.50

number\_of\_coffees = 4

# Prints "6.0"

print(coffee\_price \* number\_of\_coffees)

# Prints "1.5"

print(coffee\_price)

# Prints "4"

print(number\_of\_coffees)

# Updating the price

coffee\_price = 2.00

# Prints "8.0"

print(coffee\_price \* number\_of\_coffees)

# Prints "2.0"

print(coffee\_price)

# Prints "4"

print(number\_of\_coffees)

Here is an example of performing arithmetic operations on variables.

Notice that when the coffee\_price variable is changed the arithmetic operation will produce a different result from earlier.

**Exponents**

Python can perform exponentiation. For example 4^3 = 4 \* 4 \* 4.

We do this with the following notation; \*\* .

# 2 to the 10th power, or 1024

print(2 \*\* 10)

# 8 squared, or 64

print(8 \*\* 2)

# 9 \* 9 \* 9, 9 cubed, or 729

print(9 \*\* 3)

# We can even perform fractional exponents

# 4 to the half power, or 2

print(4 \*\* 0.5)

**Modulo**

Python offers a companion to the division operator called the modulo. The modulo % operator gives the remainder of a division calculation. If the number is divisible, then the result of the modulo operator will be 0.

# Prints 4 because 29 / 5 is 5 with a remainder of 4

print(29 % 5)

# Prints 2 because 32 / 3 is 10 with a remainder of 2

print(32 % 3)

# Modulo by 2 returns 0 for even numbers and 1 for odd numbers

# Prints 0

print(44 % 2)

The modulo operator is useful in programming when we want to perform an action every nth-time the code is run.

**Concatenation**

The + operator can also be used to add two strings together.

greeting\_text = "Hey there!"

question\_text = "How are you doing?"

full\_text = greeting\_text + question\_text

# Prints "Hey there!How are you doing?"

print(full\_text)

full\_text = greeting\_text + " " + question\_text

# Prints "Hey there! How are you doing?"

print(full\_text)

If you want to concatenate a string with a number, you will need to make the number a string first. We use the str() function. If you are trying to print() a numeric variable you can use commas to pass it as a different argument rather than converting it to a string.

birthday\_string = "I am "

age = 10

birthday\_string\_2 = " years old today!"

# Concatenating an integer with strings is possible if we turn the integer into a string first

full\_birthday\_string = birthday\_string + str(age) + birthday\_string\_2

# Prints "I am 10 years old today!"

print(full\_birthday\_string)

# If we just want to print an integer

# we can pass a variable as an argument to

# print() regardless of whether

# it is a string.

# This also prints "I am 10 years old today!"

print(birthday\_string, age, birthday\_string\_2)

**Plus Equals +=**

We can use the shorthand for updating variables. When you have a number saved in a variable and want to add to the current value of the variable we can use +=.

# First we have a variable with a number saved

number\_of\_miles\_hiked = 12

# Then we need to update that variable

# Let's say we hike another two miles today

number\_of\_miles\_hiked += 2

# The new value is the old value

# Plus the number after the plus-equals

print(number\_of\_miles\_hiked)

# Prints 14

Here is an example of this in action. Notice that we are NOT re-assigning the variable to the number 2. Instead we are adding the value 2 to the original value.

The += can also be used for string concatenation!

hike\_caption = "What an amazing time to walk through nature!"

# Almost forgot the hashtags!

hike\_caption += " #nofilter"

hike\_caption += " #blessed"

What an amazing time to walk through nature! #nofilter #blessed

**Multi-line Strings**

Python strings are very flexible. However, there maybe instances where we wish to print a string over multiple lines. In order to do this we use triple quotation marks;

leaves\_of\_grass = """

And here is an example of a multi-line string. A string that takes up multiple lines. What if we wanted to provide a quotation within our string? My dog dylan says; 'Woof, woof, woof!'. The tripple quotation marks prevent us from closing the string too early!

"""

**Review**

my\_age = 35

half\_my\_age = 35/2

greeting = "Hi! Nice to meet you."

name = "Simon"

greeting\_with\_name = greeting + " " + "My name is " + name + " and I am " + str(my\_age) + "."

print(greeting\_with\_name)

#Prints: Hi! Nice to meet you. My name is Simon and I am 35.

**User Input**

Often we want a user of a program to enter new information.

We can assign information to a variable using input(). The input() function requires a prompt message, which it will print out for the user before they enter new information.

likes\_snakes = input("Do you like snakes? ")

In this message above the following would occur:

1. The program would print ‘Do you like snakes?’ for the user.
2. The user would enter an answer (e.g. ‘Yes! I have a python) and press ENTER.
3. The variable likes\_snakes would be assigned a value of the user’s answer

Not only can input() be used to collect information, but you can store the information and use it to simulate an interaction.

>>> favorite\_fruit = input("What is your favorite fruit? ")

What is your favorite fruit? mango

>>> print("Oh cool! I like " + favorite\_fruit + " too, but I think my favorite fruit is apple.")

Oh cool! I like mango too, but I think my favorite fruit is apple.

**Control Flow**

A picture containing screenshot, diagram, line, plan

Description automatically generated

You wake up in the morning. Is it a weekend or a weekday? Is it raining or is it not?

This simple example is a demonstration of how control flow works. The answers to these questions control the flow of your morning. Python will execute script from the top down until there is nothing left to run.

It is your job to include gateways, known as conditional statements, to tell the computer when it should execute certain blocks of code or not.

**Boolean Expressions**

In order to build control flow into our program, we may wish to check if something is true or not. A Boolean can either be true or false.

If we take the example from above; today is a weekday is a Boolean. It is either true, or false.

**Relational Operators: Equals and Not Equals**

Relational operators compare two items and return either True or False.

The two relational operators we’ll cover are;

* Equals: ==
* Not Equals: !=

These operators compare two items and return True or False depending on whether they are equal or not.

1 == 1 # True

2 != 4 # True

3 == 5 # False

'7' == 7 # False

In the last example, ‘7’ is a string and 7 is a number. Therefore the relational operator will evaluate to false.

**Boolean Variables**

When we type True and False into the code editor, they appear in different colour to variables or strings. This is because they are their own special type: bool.

You can set a variable equal to a Boolean expression.

bool\_one = 5 != 7 #True

bool\_two = 1 + 1 != 2 #False

bool\_three = 3 \* 3 == 9 #True

If we printed we would get;

print(bool\_one) # True

print(bool\_two) # False

print(bool\_three) # True

Remember that Boolean variables must always be in capital letters. If you use quotations you will get a type of str.

my\_baby\_bool = "true"

print(type(my\_baby\_bool))

#Prints: <class 'str'>

my\_baby\_bool\_two = True

print(type(my\_baby\_bool\_two))

#Prints: <class 'bool'>

**If Statement**

An if statement dictates what code should be run if a condition is evaluated to True (or False).

if 2 == 4 - 2:

print("apple")

In this instance, ‘apple’ would be printed to the console because 2 == 4-2.

Here are some simple if statements that checks a username;

# Enter a user name here, make sure to make it a string

user\_name = "angela\_catlady\_87"

if user\_name == "Dave":

print("Get off my computer Dave!")

if user\_name == "angela\_catlady\_87":

print("I know it is you, Dave! Go away!")

**Relational Operators II**

* > greater than
* >= greater than or equal to
* < less than
* <= less than or equal to

**Boolean Operators: and**

Sometimes you may need more than one Boolean expression to cover. In these cases you can build larger Boolean expressions using Boolean operators. These operators (also known as logical operators) combine smaller Boolean expressions into larger Boolean expressions.

* and
* or
* not

(1 + 1 == 2) and (2 + 2 == 4) # True

(1 > 9) and (5 != 6) # False

(1 + 1 == 2) and (2 < 1) # False

(0 == 10) and (1 + 1 == 1) # False

Here is an example of the and operator.

**Boolean Operators: or**

The Boolean operator or combines two expressions into a larger expression that is True if either component is True.

True or (3 + 4 == 7) # True

(1 - 1 == 0) or False # True

(2 < 0) or True # True

(3 == 8) or (3 > 4) # False

**Boolean Operators: not**

When applied to any Boolean expression, it reverses the Boolean value. So if we have a True statement and apply a not operator we get a False statement.

not 1 + 1 == 2 # False

not 7 < 0 # True

In these examples; 1 + 1 does equal 2. The not operator inverses True to False.

7 is not less than 0 so this is False. The not operator inverses False to True.

statement\_one = not (4 + 5 <= 9)

statement\_two = not (8 \* 2) != 20 - 4

credits = 120

gpa = 1.8

if not(credits >= 120):

print("You do not have enough credits to graduate.")

if not(gpa >= 2.0):

print("Your GPA is not high enough to graduate.")

if not(credits >= 120) and not ( gpa >= 2.0):

print("You do not meet either requirement to graduate!")

If your credits are not greater than or equal to 120 a message is printed.

If your gpa is not greater than or equal to 2.0 a message is printed.

If credits are not greater than or equal to 120 AND gpa is not greater than or equal to 2.0 a message is printed to the console.

**Else Statements**

Else statements allow us to describe what we want to happen when a condition is not met. Else statements appear in conjunction with if statements. For example;

if weekday:

print("wake up at 6:30")

else:

print("sleep in")

On the weekend ‘sleep in’ is printed.

On the weekday ‘wake up at 6:30’ is printed.

**Else If Statements**

We also have elif statements.

We can use elif statements to control the order we want our program to check each of our conditional statements. First the if statement is checked then each elif statement is checked from top to bottom. Finally the else code is executed if none of the previous conditions have been met.

print("Thank you for the donation!")

if donation >= 1000:

print("You've achieved platinum status")

elif donation >= 500:

print("You've achieved gold donor status")

elif donation >= 100:

print("You've achieved silver donor status")

else:

print("You've achieved bronze donor status")

If all of the elif statements were simple if statements, then the first three messages would print as all the conditions had been met.

**Review**

* Boolean expressions are statements that can be either True or False
* A boolean variable is a variable that is set to either True or False.
* We can create boolean expressions using relational operators:
  + ==: Equals
  + !=: Not equals
  + >: Greater than
  + >=: Greater than or equal to
  + <: Less than
  + <=: Less than or equal to
* if statements can be used to create control flow in your code.
* else statements can be used to execute code when the conditions of an if statement are not met.
* elif statements can be used to build additional checks into your if statements