PYTHON CHEAT SHEET

100 DAYS OF CODE
COMPLETE PROFESSIONAL
PYTHON BOOTCAMP

BASICS

Print

Prints a string into the console.

```
print("Hello World")
```

Input

Prints a string into the console,
and asks the user for a string input.

```
input("What's your name")
```

Comments

Adding a # symbol in font of text
lets you make comments on a line of code.
The computer will ignore your comments.

```
#This is a comment
print("This is code")
```

Variables

A variable give a name to a piece of data.
Like a box with a label, it tells you what's
inside the box.

```
my_name = "Angela"
my_age = 12
```

The += Operator

This is a convient way of saying: "take the
previous value and add to it.

```
my_age = 12
my_age += 4
#my_age is now 16
```

PYTHON CHEAT SHEET

100 DAYS OF CODE
COMPLETE PROFESSIONAL
PYTHON BOOTCAMP

DATA TYPES

Integers

Integers are whole numbers.

```
my_number = 354
```

Floating Point Numbers

Floats are numbers with decimal places. When you do a calculation that results in a fraction e.g. 4 ÷ 3 the result will always be a floating point number.

```
my_float = 3.14159
```

Strings

A string is just a string of characters. It should be surrounded by double quotes.

```
my_string = "Hello"
```

String Concatenation

You can add strings to string to create a new string. This is called concatenation. It results in a new string.

```
"Hello" + "Angela"
#becomes "HelloAngela"
```

Escaping a String

Because the double quote is special, it denotes a string, if you want to use it in a string, you need to escape it with a "\"

```
speech = "She said: \"Hi\""
print(speech)
#prints: She said: "Hi"
```

PYTHON CHEAT SHEET

100 DAYS OF CODE
COMPLETE PROFESSIONAL
PYTHON BOOTCAMP

F-Strings

You can insert a variable into a string using f-strings.
The syntax is simple, just insert the variable in-between a set of curly braces {}.

```
days = 365
print(f"There are {days} in a year")
```

Converting Data Types

You can convert a variable from 1 data type to another.
Converting to float:
`float()`
Converting to int:
`int()`
Converting to string:
`str()`

```
n = 354
new_n = float(n)
print(new_n) #result 354.0
```

Checking Data Types

You can use the type() function to check what is the data type of a particular variable.

```
n = 3.14159
type(n) #result float
```

PYTHON CHEAT SHEET

100 DAYS OF CODE
COMPLETE PROFESSIONAL
PYTHON BOOTCAMP

MATHS

Arithmetic Operators

You can do mathematical calculations with Python as long as you know the right operators.

```
3+2 #Add
4-1 #Subtract
2*3 #Multiply
5/2 #Divide
5**2 #Exponent
```

The += Operator

This is a convenient way to modify a variable. It takes the existing value in a variable and adds to it.
You can also use any of the other mathematical operators e.g. -= or *=

```
my_number = 4
my_number += 2
#result is 6
```

The Modulo Operator

Often you'll want to know what is the remainder after a division.
e.g. 4 ÷ 2 = 2 with no remainder
but 5 ÷ 2 = 2 with 1 remainder
The modulo does not give you the result of the division, just the remainder.
It can be really helpful in certain situations, e.g. figuring out if a number is odd or even.

```
5 % 2
#result is 1
```

PYTHON CHEAT SHEET

100 DAYS OF CODE
COMPLETE PROFESSIONAL
PYTHON BOOTCAMP

ERRORS

Syntax Error

Syntax errors happen when your code does not make any sense to the computer. This can happen because you've misspelt something or there's too many brackets or a missing comma.

```
print(12 + 4))
  File "<stdin>", line 1
    print(12 + 4))
                  ^
SyntaxError: unmatched ')'
```

Name Error

This happens when there is a variable with a name that the computer does not recognise. It's usually because you've misspelt the name of a variable you created earlier.
Note: variable names are case sensitive!

```
my_number = 4
my_Number + 2
Traceback (most recent call
last): File "<stdin>", line 1,
NameError: name 'my_Number'
is not defined
```

Zero Division Error

This happens when you try to divide by zero, This is something that is mathematically impossible so Python will also complain.

```
5 % 0
Traceback (most recent call
last): File "<stdin>", line 1,
ZeroDivisionError: integer
division or modulo by zero
```

PYTHON CHEAT SHEET

100 DAYS OF CODE
COMPLETE PROFESSIONAL
PYTHON BOOTCAMP

FUNCTIONS

Creating Functions

This is the basic syntax for a function in Python. It allows you to give a set of instructions a name, so you can trigger it multiple times without having to re-write or copy-paste it. The contents of the function must be indented to signal that it's inside.

```
def my_function():
    print("Hello")
    name = input("Your name:")
    print("Hello")
```

Calling Functions

You activate the function by calling it. This is simply done by writing the name of the function followed by a set of round brackets. This allows you to determine when to trigger the function and how many times.

```
my_function()
my_function()
#The function my_function
#will run twice.
```

Functions with Inputs

In addition to simple functions, you can give the function an input, this way, each time the function can do something different depending on the input. It makes your function more useful and re-usable.

```
def add(n1, n2):
    print(n1 + n2)

add(2, 3)
```

PYTHON CHEAT SHEET

100 DAYS OF CODE
COMPLETE PROFESSIONAL
PYTHON BOOTCAMP

Functions with Outputs

In addition to inputs, a function can also have an output. The output value is proceeded by the keyword "return".
This allows you to store the result from a function.

```
def add(n1, n2):
    return n1 + n2

result = add(2, 3)
```

Variable Scope

Variables created inside a function are destroyed once the function has executed.
The location (line of code) that you use a variable will determine its value.
Here n is 2 but inside my_function() n is 3.
So printing n inside and outside the function will determine its value.

```
n = 2
def my_function():
    n = 3
    print(n)

print(n) #Prints 2
my_function() #Prints 3
```

Keyword Arguments

When calling a function, you can provide a keyword argument or simply just the value.
Using a keyword argument means that you don't have to follow any order when providing the inputs.

```
def divide(n1, n2):
    result = n1 / n2
#Option 1:
divide(10, 5)
#Option 2:
divide(n2=5, n1=10)
```

PYTHON CHEAT SHEET

100 DAYS OF CODE
COMPLETE PROFESSIONAL
PYTHON BOOTCAMP

CONDITIONALS

If

This is the basic syntax to test if a condition is true. If so, the indented code will be executed, if not it will be skipped.

```
n = 5
if n > 2:
   print("Larger than 2")
```

Else

This is a way to specify some code that will be executed if a condition is false.

```
age = 18
if age > 16:
   print("Can drive")
else:
   print("Don't drive")
```

Elif

In addition to the initial If statement condition, you can add extra conditions to test if the first condition is false.
Once an elif condition is true, the rest of the elif conditions are no longer checked and are skipped.

```
weather = "sunny"
if weather == "rain":
   print("bring umbrella")
elif weather == "sunny":
   print("bring sunglasses")
elif weather == "snow":
   print("bring gloves")
```

PYTHON CHEAT SHEET

100 DAYS OF CODE
COMPLETE PROFESSIONAL
PYTHON BOOTCAMP

and

This expects both conditions either side of the and to be true.

```
s = 58
if s < 60 and s > 50:
   print("Your grade is C")
```

or

This expects either of the conditions either side of the or to be true. Basically, both conditions cannot be false.

```
age = 12
if age < 16 or age > 200:
   print("Can't drive")
```

not

This will flip the original result of the condition. e.g. if it was true then it's now false.

```
if not 3 > 1:
   print("something")
   #Will not be printed.
```

comparison operators

These mathematical comparison operators allow you to refine your conditional checks.

```
> Greater than
< Lesser than
>= Greater than or equal to
<= Lesser than or equal to
== Is equal to
!= Is not equal to
```

PYTHON CHEAT SHEET

100 DAYS OF CODE
COMPLETE PROFESSIONAL
PYTHON BOOTCAMP

LOOPS

While Loop

This is a loop that will keep repeating itself until the while condition becomes false.

```
n = 1
while n < 100:
   n += 1
```

For Loop

For loops give you more control than while loops. You can loop through anything that is iterable. e.g. a range, a list, a dictionary or tuple.

```
all_fruits = ["apple",
   "banana", "orange"]
for fruit in all_fruits:
   print(fruit)
```

_ in a For Loop

If the value your for loop is iterating through, e.g. the number in the range, or the item in the list is not needed, you can replace it with an underscore.

```
for _ in range(100):
   #Do something 100 times.
```

break

This keyword allows you to break free of the loop. You can use it in a for or while loop.

```
scores = [34, 67, 99, 105]
for s in scores:
   if s > 100:
      print("Invalid")
      break
   print(s)
```

continue

This keyword allows you to skip this iteration of the loop and go to the next. The loop will still continue, but it will start from the top.

```
n = 1
while n < 100:
   if n % 2 == 0:
      continue
   print(n)
#Prints all the odd numbers
```

Infinite Loops

Sometimes, the condition you are checking to see if the loop should continue never becomes false. In this case, the loop will continue for eternity (or until your computer stops it). This is more common with while loops.

```
while 5 > 1:
   print("I'm a survivor")
```

PYTHON CHEAT SHEET

100 DAYS OF CODE
COMPLETE PROFESSIONAL
PYTHON BOOTCAMP

LIST METHODS

Adding Lists Together

You can extend a list with another list by using the extend keyword, or the + symbol.

```
list1 = [1, 2, 3]
list2 = [9, 8, 7]
new_list = list1 + list2
list1 += list2
```

Adding an Item to a List

If you just want to add a single item to a list, you need to use the .append() method.

```
all_fruits = ["apple",
   "banana", "orange"]
all_fruits.append("pear")
```

List Index

To get hold of a particular item from a list you can use its index number.
This number can also be negative, if you want to start counting from the end of the list.

```
letters = ["a", "b", "c"]
letters[0]
#Result:"a"
letters[-1]
#Result: "c"
```

List Slicing

Using the list index and the colon symbol you can slice up a list to get only the portion you want.
Start is included, but end is not.

```
#list[start:end]
letters = ["a","b","c","d"]
letters[1:3]
#Result: ["b", "c"]
```

PYTHON CHEAT SHEET

100 DAYS OF CODE
COMPLETE PROFESSIONAL
PYTHON BOOTCAMP

BUILT IN FUNCTIONS

Range

Often you will want to generate a range of numbers. You can specify the start, end and step.
Start is included, but end is excluded:
start >= range < end

```
# range(start, end, step)
for i in range(6, 0, -2):
   print(i)

# result: 6, 4, 2
# 0 is not included.
```

Randomisation

The random functions come from the random module which needs to be imported.
In this case, the start and end are both included
start <= randint <= end

```
import random
# randint(start, end)
n = random.randint(2, 5)
#n can be 2, 3, 4 or 5.
```

Round

This does a mathematical round.
So 3.1 becomes 3, 4.5 becomes 5 and 5.8 becomes 6.

```
round(4.6)
# result 5
```

abs

This returns the absolute value.
Basically removing any -ve signs.

```
abs(-4.6)
# result 4.6
```

PYTHON CHEAT SHEET

100 DAYS OF CODE
COMPLETE PROFESSIONAL
PYTHON BOOTCAMP

MODULES

Importing

Some modules are pre-installed with python e.g. random/datetime
Other modules need to be installed from pypi.org

```
import random
n = random.randint(3, 10)
```

Aliasing

You can use the as keyword to give your module a different name.

```
import random as r
n = r.randint(1, 5)
```

Importing from modules

You can import a specific thing from a module. e.g. a function/class/constant
You do this with the from keyword.
It can save you from having to type the same thing many times.

```
from random import randint
n = randint(1, 5)
```

Importing Everything

You can use the wildcard (*) to import everything from a module. Beware, this usually reduces code readability.

```
from random import *
list = [1, 2, 3]
choice(list)
# More readable/understood
#random.choice(list)
```

PYTHON CHEAT SHEET

100 DAYS OF CODE
COMPLETE PROFESSIONAL
PYTHON BOOTCAMP

CLASSES & OBJECTS

Creating a Python Class

You create a class using the class keyword.
Note, class names in Python are PascalCased.
So to create an empty class □

```
class MyClass:
    #define class
```

Creating an Object from a Class

You can create a new instance of an object by using the class name + ()

```
class Car:
    pass

my_toyota = Car()
```

Class Methods

You can create a function that belongs to a class, this is known as a method.

```
class Car:
    def drive(self):
        print("move")
my_honda = Car()
my_honda.drive()
```

Class Variables

You can create a varaiable in a class.
The value of the variable will be available to all objects created from the class.

```
class Car:
    colour = "black"
car1 = Car()
print(car1.colour) #black
```

PYTHON CHEAT SHEET

100 DAYS OF CODE
COMPLETE PROFESSIONAL
PYTHON BOOTCAMP

The __init__method

The init method is called every time a new object is created from the class.

```
class Car:
    def __init__(self):
        print("Building car")
my_toyota = Car()
#You will see "building car"
#printed.
```

Class Properties

You can create a variable in the init() of a class so that all objects created from the class has access to that variable.

```
class Car:
    def __init__(self, name):
        self.name = "Jimmy"
```

Class Inheritance

When you create a new class, you can inherit the methods and properties of another class.

```
class Animal:
    def breathe(self):
        print("breathing")
class Fish(Animal):
    def breathe(self):
        super.breathe()
        print("underwater")
nemo = Fish()
nemo.breathe()
#Result:
#breathing
#underwater
```