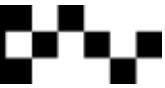
Software Development

Learning Python 3

This document will introduce you to programming in Python.

Units 3 & 4

Contents



What is Python?

Python is a programming language that was first created by Guido von Rossum in 1990 and was named after the British comedy Monty Python. Since then it has been developed by a large team of volunteers around the world and released under the Creative Commons License so that it is freely distributed and used without having to pay a license fee.

Python is a general purpose programming language that can be used on any computer with any operating system. Computers with Unix, Linux, Sun, Oracle or Apple operating systems have Python installed with the initial OS installation, computers with the Windows OS installed must download and install Python from www.python.org.

Who uses Python?

Short answer: nearly everyone.

- Google uses Python in its web search system and employ's Pythons creator.
- Youtube video sharing is written in Python.
- BitTorrent, the peer-to-peer file sharing system is written in Python.
- Intel, Cisco, Hewlett-Packard, Seagate, Qualcomm and IBM use Python for hardware testing.
- Industrial Light & Magic, Pixar, Disney and others use Python in the production of animated movies.
- NASA, Los Alamos, Fermilab, JPL and other use Python for scientific programs.
- New York Stock Exchange uses Python for financial marketing.
- iRobot uses Python to program its robotic vacuum cleaners.
- And plenty more…

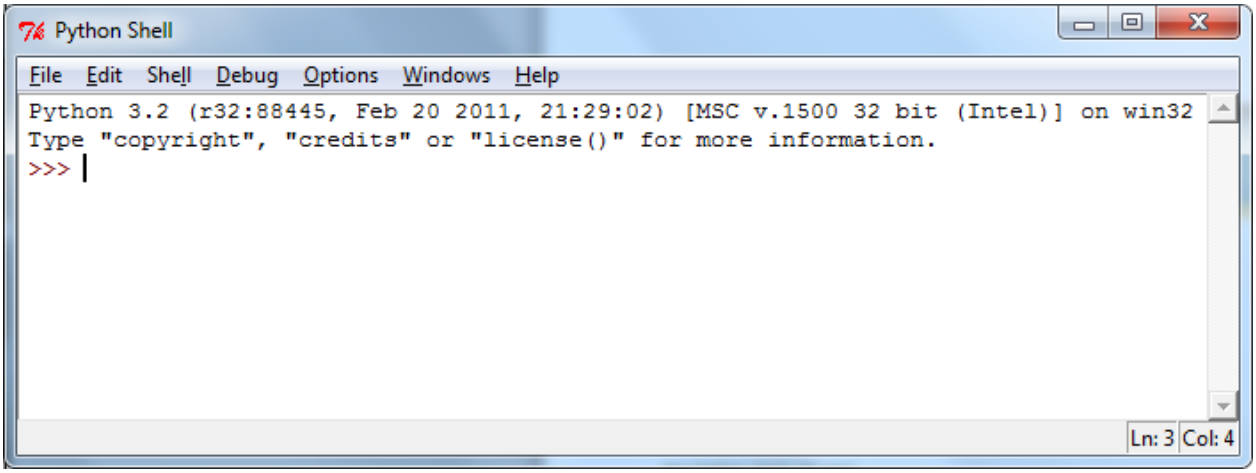
What can you do with Python?

Python is a well-designed programming language that is useful for accomplishing real-world tasks, the sorts of things that software developers do every day. You can use Python for everything from web site development, robotics to 3D modelling, gaming and spacecraft control. The limit is endless.

Python is easy to learn, and easier to use compared to other programming languages. You can expect to be programming in a couple of hours.

Starting Python

Once Python is installed open **IDLE (Python GUI)**



The Python Shell will open.

The information on the top line shows the version of Python that is installed, when IDLE was built, the processor and the current operating system of the computer that is running IDLE.

The Python Shell is where you can explore Python syntax, get interactive help on commands, and debug short programs. The graphical Python Shell (named IDLE) also contains a text editor that supports Python syntax colouring and integrates with the Python Shell.

Data Types

There are several different data types. Each store, behave or react in a particular way. In this chapter we will have a look at each one.

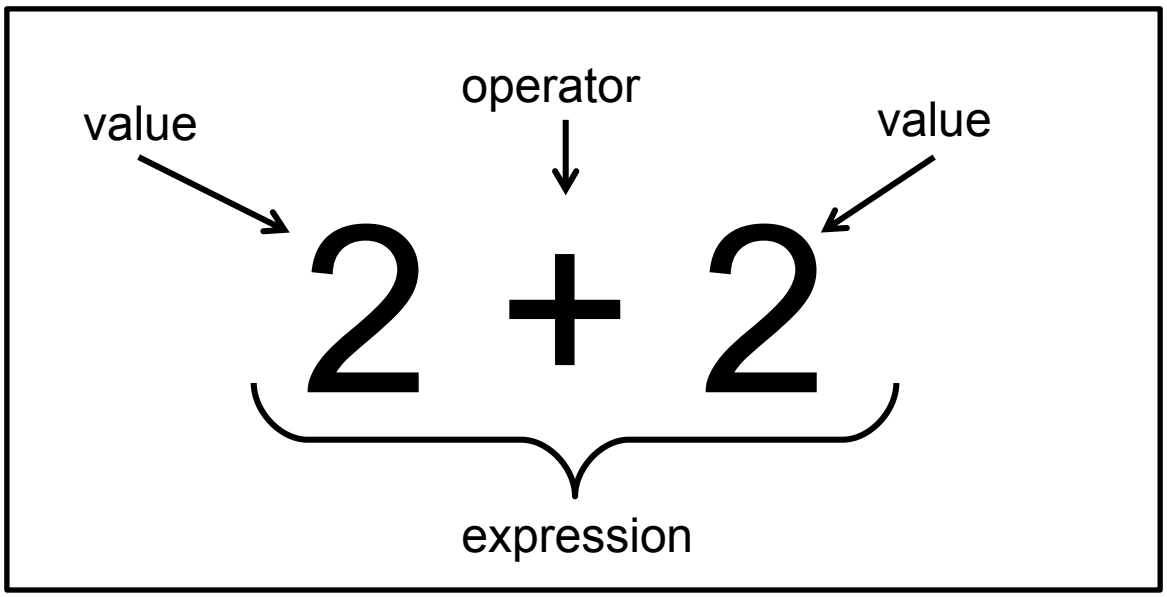
Data types are:

- ✓ Integers
- ✓ Strings
- ✓ Booleans
- ✓ Arrays

Python's Shell

Python's shell can be used as a calculator; on the following pages you will be using it to calculate several mathematical expressions.

Expressions



Integers

Integers are **whole** numbers.

34 788 9 005

IDLE can be used as a calculator.

This is a quick way to manipulate and calculate numbers.

Python uses these operators to manipulate integers (numbers).

+ addition

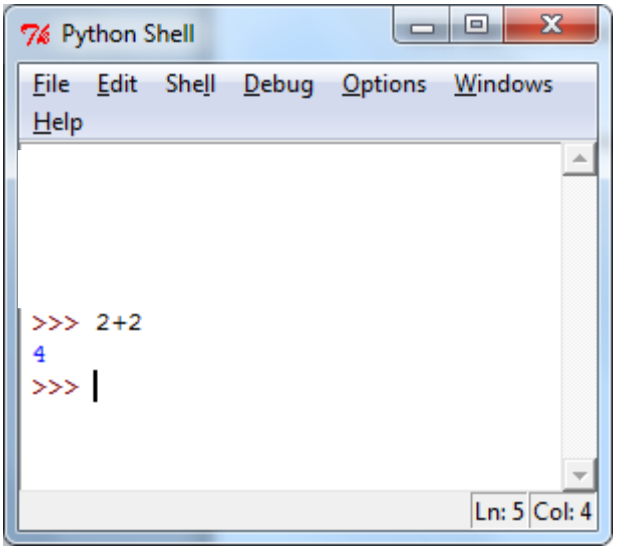
- minus

* multiply

/ divide

% remainder

** power



In IDLE

Use IDLE to answer the following equations:

>>> 4+17

>>> -9+4-3

>>> 9*8 - -4

>>> 7+9*2

>>> 20%

>>> 64**4

>>> 15*90*2

>>> 2*(3+4)

>>> 2+18/4

>>> 6+2*7/3+5

>>> 7**3

>>> 6**10/5%4

Floating Point Numbers

Floating point numbers are fractions or decimals.

56.03 44.9 0.99

Python uses these operators to manipulate integers (numbers).

+ addition

- minus

* multiply

/ divide

% remainder

** power

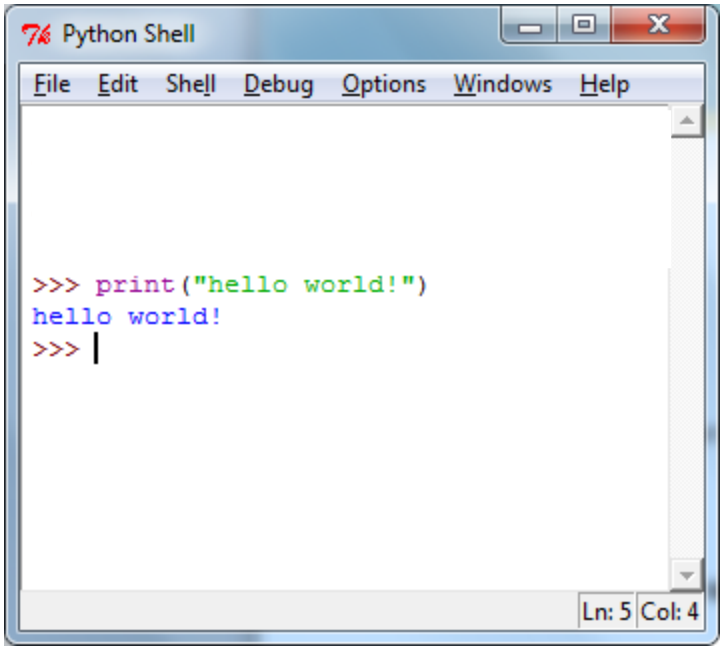
In IDLE

Use IDLE to answer the following equations:

```
>>> 1/3

>>> 12+4/13

>>> -9+4-3

>>> 2*(3+4)

>>> 9*8 - -4

>>> 2+18/4

>>> 7+9*2

>>>  6+2*7/3+5

>>> 20%17
```

Strings and Characters

Stings are words or sentences and characters are single letters. In python we call both characters and sentences strings.



Python has built-in functions that perform particular things.

The print() function prints whatever is written between the parenthesis to the screen.

All built-in functions look the same.

functionName()

The only difference is what the function does.

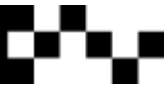
The quote " " marks in the print statement mean that the information is read as a string.

Strings are text composed of any ASCII characters (alphabetic letters, numerals, punctuation and control codes like carriage return, backspace, bell) up to about 2 billion characters. While it can store numerals, it cannot interpret the value of numbers and instead treats them as plain text. So if you add "1" and "2" you'll get "12" instead of 3!

In IDLE

Use IDLE to print the following string statements:

```
>>> print("hello")
>>> print(2*15)
>>> print("The quick brown fox jumped over the lazy dog")
>>> print("My birthday is : 00/00/00")
>>> print(“I am learning to program in Python. I am studying Software Development”)
```



Booleans

Booleans have two states, they are either True or False.

In IDLE

Use IDLE to print the following string statements:

```
>>> x = set('abcde')
>>> y= set('zyxwv')
>>> x
```

set() regards the elements in the parenthesis as iterable data.

What does IDLE print?

```
>>> 'e' in x
```

type the ' ' around the letter you are searching for or you'll get an error

What does IDLE print?

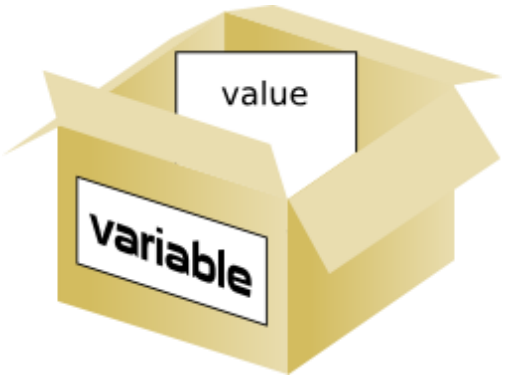
```
>>>  'r' in y
```

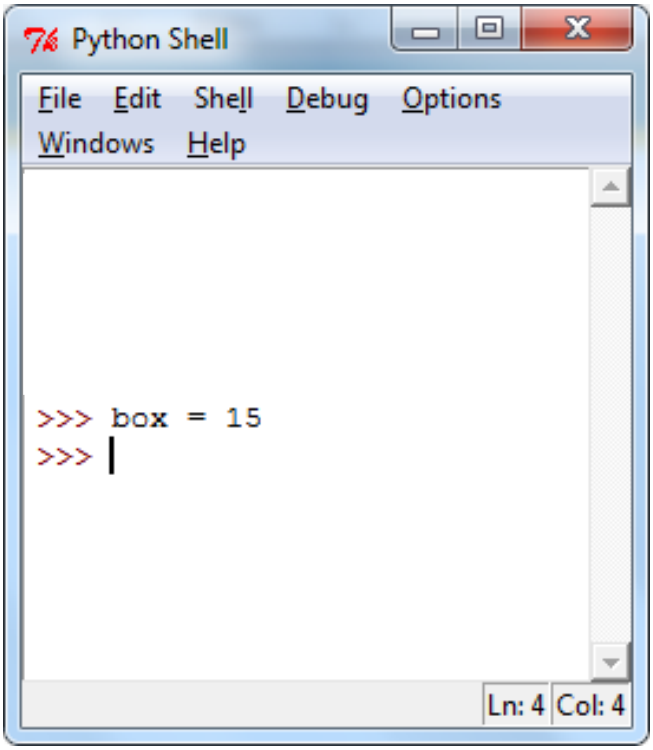
What does IDLE print?

Variables

Variables are some of the most important pieces of programming code.

Variables are like boxes. They store values.



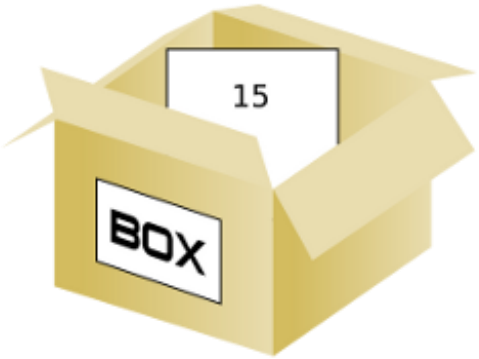


Type >>>**box = 15** into IDLE.

When you press Enter, nothing will happen.

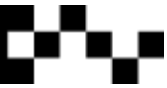
Python has stored the variable **box** with the value of **15** so that it is able to use it later in the program.

Now type >>>**box**.



The equals sign (=) is what's called an **assignment** operator. You can store values inside variables with the **=** sign.

Now **box** is now worth **15**.



Adding to Variables

Type
>>> box + 5
Press ENTER
So, that is like writing 15 + 5

What is the response IDLE prints?

>>>

Changing Values

The first time you store a value inside a variable, Python will create that variable. Each time after that, the value will only be replaced.

```
>>> box = 15
>>> box + 5
20
>>> box = 3
>>> box + 5
8
>>>
```

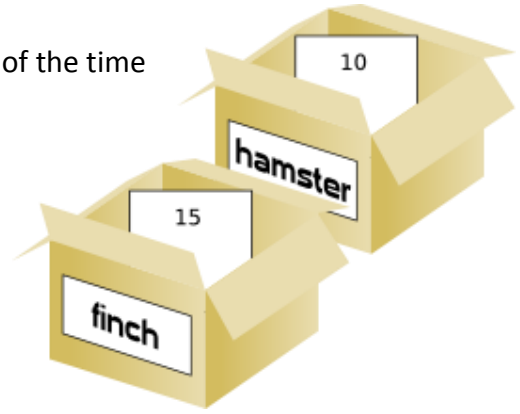
Evaluating Variables

A variable is only a name for a value, we write expressions with variables like this:

```
>>> box = 15
>>> box + box
30
>>> box - box
0
>>>
```

More than one Variable

We are not limited to using only one variable. Most of the time we need to use multiple variables.



>>> hamster = 10
>>> finch = 15

Changing Variables

Let's change the value in the **box** variable to:

>>> box = finch + hamster
>>> box
25
>>>

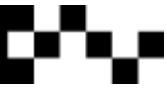
In IDLE

>>> tree = “elm”
>>> sky = “blue”
>>> animal = “dog”
>>> tree, sky, animal If you place a comma between variable names it prints them on a single line.

>>> one = 1
>>> two = 2
>>> three = 3
>>> tree, one, sky, two, animal, three

>>>sum = one + two
>>> sum
>>>total = animal + sky + tree
>>> total

Explain what is happens in all four sections in the above script.



input()

input() is an inbuilt function* that lets the user input information into the program. Usually input() is stored in a variable so the information can be stored.

input() is not a data type, but it used to store them.

Variables and input()

Type the following line of code into the Python shell:

word = input()

It should look like this:

```
>>>
>>> word = input()
```

After you press enter, the program will wait for input.

Type anything into IDLE, press enter, nothing will happen. You will get:

>>>

Python has now stored what you typed into the variable called **word**.

>>> print(word)

*In a later section we will discuss functions further.

Hello World

Python has more functionality than just IDLE.

In the Python Shell, click on File> New Window (or press Ctrl+N)

This will open a script document.

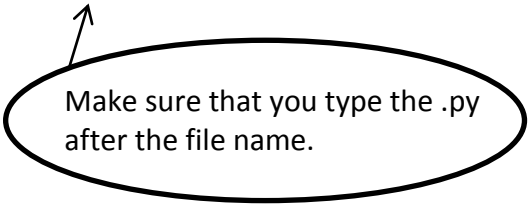
Type this into the document:

```
print('Hello world!')
print('What is your name?')
myName = input()
print('It is good to meet you, ' + myName)
```

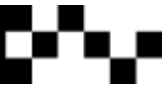
Going through line by line:

print('Hello world!')
Calls the print function and displays 'Hello world!'
print('What is your name?')
Calls the print function and displays 'What is your name?'
myName = input()
Creates a variable called myName, and asks for user input through the input function
print('It is good to meet you, ' + myName)
Calls the print function and displays 'It's good to meet you,' and prints the input that was stored within the variable *myName*.

To run the script, it must be saved first.
Create a new folder called **Scripts**, name the file **helloworld.py**, then hit the **Save** button.



Press F5 to run the script.



Task 1: Inspector.py

Turn this pseudocode into Python script:

yourName ← empty string

PRINT 'Enter your name: '
INPUT ← yourName

PRINT ‘Welcome Inspector ' and print Inspectors name

PRINT ‘You have been sent on a mission to find why so many people have gone missing. Investigations have lead you to the Horror Mansion on Black Moon Lane.'

PRINT 'We wish you the best of luck.'

PRINT '...'

PRINT 'You have just arrived at Horror Mansion.’
PRINT ‘It is a large dark spooky house with strange noises coming from inside.’
PRINT ‘You hop out of the car and walk closer...’
PRINT 'You tread carefully onto the rotten wooden porch.’
PRINT ‘You trip over a loose plank of wood and fall over.
PRINT ‘Your knee hurts. But you notice something under the porch...'
PRINT 'It is a box...'
PRINT ()

Continue the story, try to put more user input into your part of the game.

Text based addition calculator

For this task, you need to create an addition calculator by using the skills you have learned. You need to be able to get the program to recognise two integers input by the user, the program stores the two integers then one integer is to be added to the other and the result should be displayed.

What does an addition calculator do?

Adds two numbers together

So, you've said, the addition calculator will add two numbers together. Is it that simple?

Where do the numbers come from? How are they stored? What happens to them when they are stored? How will it all be displayed? How do you ask for user input?

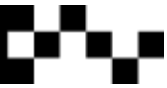
Think logically, how would you get your addition calculator to do this? Write pseudocode to help you think through this problem.

HINT:

Step 1: User inputs one number
Step 2: User inputs second number
Step 3: Program adds numbers together
Step 4: Program displays answer

You will need three variables, what should they be called?

Number 1, number 2, answer



Test your theories in Python IDLE and see what you can do. Save it as **calculator.py**.
You will have something like:

```
print('Choose a number')
number1 = input()

print('Choose another number')
number2 = input()

print ('Now they will be added together.')
answer = number1 + number 2

print('The answer is: ' + answer)
```

But it doesn't work, you will see that the two numbers are not added together.

Python is a strongly typed language

Some languages are **strongly typed** and others are **weakly typed**.

In a strongly typed language, all variables must have their type declared before values can be stored in them. Such languages are strict about exchanging values between variables of different types.

Weakly typed languages don't force programmers to declare variables beforehand, and they **infer** (decide on by guessing) the type of a variable by how it's being treated. .e.g. `C = "Cat"` would treat variable C as a text type because it's being forced to contain a string.

Some languages allow relatively free data exchanges (e.g. C does not mind if you copy a value from a pointer variable into a long integer or a char to a short integer). Other languages would require the programmer to **explictly** convert from one type to another (to prevent unintentional casting) e.g. `varA = CINT(varB)` would convert varB to an integer before storing it in varA.

Implicit casting is when a value is converted from type type to another simply by assigning it e.g. `dblA = intB` converts integer *intB* to double *dblA* simply by copying it.

Languages Compared

Java, **C**, **C#**, **Pascal**, **Ada** - require all variables to have a defined type and support the use of **explicit casts** (deliberate conversions) of types.

Smalltalk, **Ruby**, **Python**, **Lisp** and **Self** are all strongly typed in that undeclared variables are prevented at runtime and they allow little implicit type conversion.

Visual BASIC is a mixed bag. It supports statically typed variables, as well as the **Variant** data type that can store data of any type. Its implicit casts are fairly liberal where, for example, one can sum string variants and pass the result into an integer.

In general, weakly typed languages offer a faster development rate at the risk of accidental improper data conversions (e.g. accidentally saying something like A=B + C and forgetting that B is a *date* type, resulting in rubbish results.

What does this all mean?

You need to declare variables at the top of your code to allow Python to read them properly.

number1= ' ' **these string marks mean that this variable is an empty string,**
number2= ' ' **meaning that it can store whatever is typed in, not just a predetermined string**

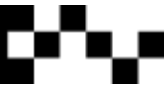
```
print('Choose a number')
number1 = input()

print('Choose another number')
number2 = input()

print ('Now they will be added together.')
answer = number1 + number 2
print('The answer is: ' + answer)
```

you don't need to declare the *answer* variable until it is used.*

Always declare variables first, before you call them in the code.

*Python reads code from the top to the bottom, so variables like *answer* are declared during the program, so they can be utilised at the correct position. Python will not understand what you want it to do with *answer* if you put it at the top because *number1* and *number2* do not have user input values yet.

It still doesn't work…



Changing Data Types

Pretend that you think like a computer.

>>> Choose a number

The user enters a **string** value of 3.

>>> Choose another number

The user enters a **string** value of 2.

Nothing happens, because strings cannot be added together. It's like saying the letter '*n*' and the letter '*z*' should be added together – it just can't be done. Only integers (numbers) can be added together to produce a digit.

How do you change a string to an integer?

answer = int(number1) + int(number2)

You have to declare the strings to behave as integers. Confused? So, what's happening is, the string in the variables are now read as integers, even though they are still strings.

Now they can be added together to make the correct answer.

But if you run the script you will get an error.

Traceback (most recent call last):
File "calculator.py", line 10, in <module>
print('The answer is: ' + answer)
TypeError: Can't convert 'int' object to str implicitly

What this error is saying is:

On line 10, print(The answer is: ' + answer) needs to be changed to a string.

Change print(The answer is: ' + answer) to:

print(The answer is: ' + ***str(answer)***)

This is so the strings that were changed into integers can now be displayed as a string.

Now your calculator should work.

IF Statements

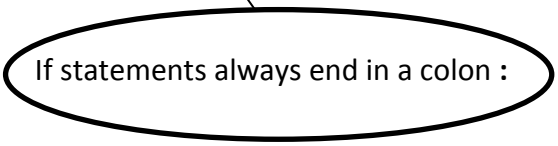
If statements are one of the most used syntaxes in any programming language.

Basically an if statement says:

If this happens do this....

Like this:

```
if  python == 6:
    print('Python has 6 letters')
```



IF ELSE Statements

IF statements can be extended to perform an else.

IF this doesn't happen,
ELSE do this instead

Like this:

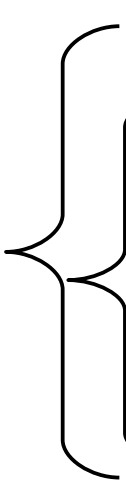
```
fish = 5

print('Type an amount of fish in the pond')
text = input()

if fish => 5:
    print('There are heaps of fish in the pond.')
else:
    print('There are not enough fish in the pond.')
```

ELIF Statements

An ELIF statement extends on an IF ELSE statement and an IF statement.

```
if horses == brown and frogs == green:
    do this...
elif horses == brown and frogs == blue:
    do this...
elif horses == white and frogs == green:
    do this...
elif horses == white and frogs == blue:
    do this...
else:
    do this...
```

Nested IF Statements



```
if trees == 12 and bugs == 100:
    print ('There are lots of bugs today')
    if worms == 3 and caterpillars == 5:
            print ('The creepy bugs are out today')
    elif worms == 2 and caterpillars == 4:
            print ('There are some creepy crawlies today')
    elif worms == 1 and caterpillars == 3:
            print ('There aren\'t many creepy bugs')
    else:
            print('There are only caterpillars today')
    elif trees <= 5 and bugs <=60:
            print('There are not enough trees for the bugs to live in')
else:
    print('You need more trees.')
```

Task 2: Calculator.py

Extending the addition calculator

Open the **calculator.py** and *Save As* **calculator-extended.py**.

The user wants to multiply, as well as add.

You will need a menu system that selects between addition or multiplication options.

You will need to use IF, ELIF and ELSE statements.

You will need to put an error message in your code, if the user selects an incorrect menu item.

You will need to create new variables to store the new inputted numbers.

EXTENSION TASK

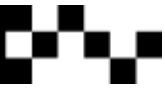
Create a fully functioning calculator that adds, subtracts, multiplies and divides.

Get the last message to print the two selected numbers and the function, e.g:

2 x 4 = 8

not *'The answer is: 8'*

HINT: you will need to concatenate variables and strings together on one line, to do this use + symbols between the variables and the strings:
print(var1 + ' + ' + var2 + ' = ' + str(answer))



Arrays

Arrays are the same as Python's Lists. In this documentation we will call them lists, but don't get confused, because they are all the same thing.

What are lists?

The list is a most versatile data type, which means it can store several different data types together in the same variable.

Lists don't need to have all the same types of values, they can have a mixture of integers, strings, Booleans and any other data type.

Creating a list is as simple as putting different comma-separated values between square brackets:

Example:

list1 = ['Rome', 'Paris', 'Melbourne', 20, 5, 72]

list2 = [1, 2, 3, 4, 5, 6, 7]

list3 = ['a', 'b', 'c', 'd', 'e']

Rules about lists

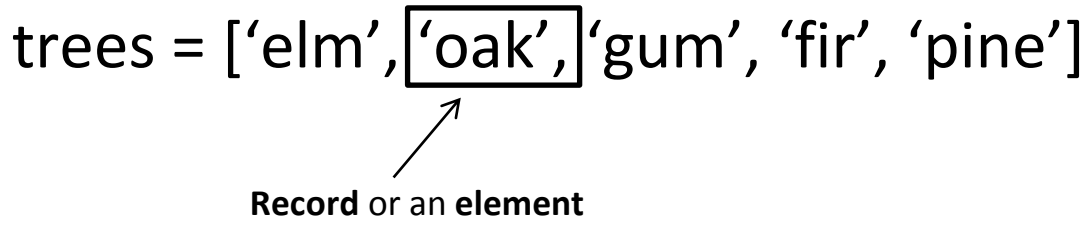
The first value in a list is always numbered 0(zero) and the following values are 1, 2, 3, 4 etc.

```
list1=['Paris', 'Rome', 'Sydney', 'New York']
          0        1        3           4
```

Computers ALWAYS start counting at zero, so knowing this helps us tell the computer which value we will be using in the list.

Records and Elements

Records and elements are the same thing, they are the list's contents, separated by commas.



Open Python Shell and type the *trees* list into IDLE.

>>> trees = ['elm', 'oak', 'gum', 'fir', 'pine']

To recall a list element type:

>>> trees[3]

This will print : fir'

Remember that it starts counting from 0 not 1.

Remember how Python displays strings and integers? Strings have ' ' around them, integers don't.

List Tasks

Convert the sentences into lists, following the example below:

Bob Smith is 42 years old, earns $30,000 a year and works in software.
bob=['Bob Smith', 42, 30000, 'Software']

Sue Jones is 45 years old and gets paid $40,000 a year, working in music.

John Black is 16years old and doesn't get paid anything. He wants to work in Information Technology.

Hint: Use all three lists: Bob, Sue and John.

What is Sue's first record, and John's third record?

Splitting Records

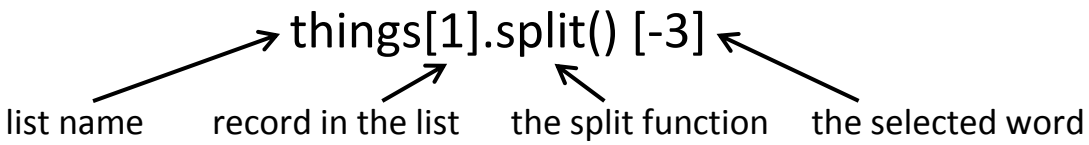
Python allows you to split string records apart. This allows you to split records that have more than one word in a string.

For example: things=['red matchbox car', 'orange and green spinning top']

I want to print the third word in the second record: '*green*'.

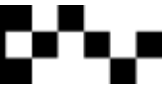
Remembering that when counting records, the first one is 0 and the second one is 1…

To print '*green*' I use the split() function.



The [-3] means that it is splitting the third-last word out of the record. When splitting records, Python counts from the last word and works backwards, that's why it's a [-3], it's taking 3 away from the last word.

What do you type to get Bob's last name to display? How about Sue's first name?



Manipulating Records

It is important to note that when using integers in a list (array), it is difficult to change one record for a period of time and then change it back while the program is running.

What is in the array stays in the array unless you physically type something else into it.

To add to an integer in the array you can do this:

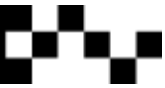
```
>>>prices=[56, 30, 50, 800]

>>>prices[2] += 9

>>>print(prices)

[56, 30, 59, 800]
```

This adds 9 to record 3 in the list. This type of digit manipulation is stored for as long as the program runs, but it doesn't update the record forever, you still have to do that if you want to keep it at 59!

Give Sue a 25% raise. How did you do it?

Give Bob a 45% deduction. How did you do it?

John finished school and went to uni, he gets paid $250 a fortnight. What do you put into the list so that you can total his annual wages? What is the total of his wages?



Database Lists

When we have two separate lists, e.g., Sue and John, we can create a single list from them by creating a new list.

people = [sue, john] note the lack of ' ' around the names, because we are calling the name of the list, not a string.

>>>print(people)

What displays when you print the two lists?

When dealing with many lists, we can collect specific records from them, to do this we type:

>>>people [1][0]

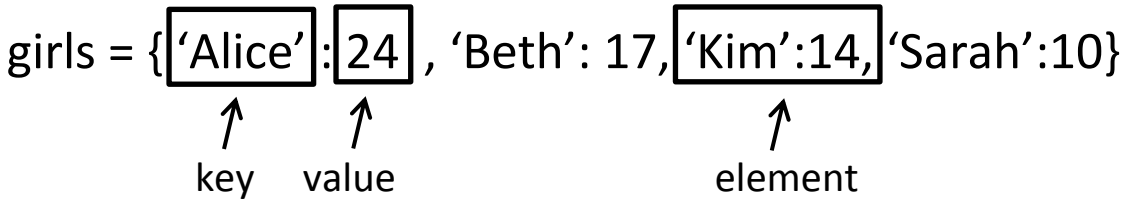
What displays when you type the above?

Python has three different types of arrays: lists, dictionaries and tuples.

What are dictionaries?

Dictionaries are made of pairs. Unlike a list, dictionaries store a key and a value in a single element.

They look like this:



Dictionary values have no restrictions. They can be made of any object (integer, string etc). However, same is not true for the keys.

There are two important points to remember about dictionary keys:

(a) More than one entry per key not allowed. Which means no duplicate key is allowed. When duplicate keys are encountered during assignment, the last assignment wins.

(b) Keys must be unchangeable. Which means you can use strings, numbers, or tuples as dictionary keys but something like ['key'] is not allowed --> because of the square brackets [ ] around the key. This will cause an error.

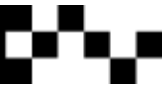
student={'Name': 'Dave', 'Age':17, 'Class': 'Software Development'}

>>>print('My name: ', student['Name'])

>>>print('Age: ', student['Age'])

>>>print(Class: ', student['Class'])

Remember, if you try to access a key that is not in the dictionary, you will get an error.



Put Sue and John's details into a dictionary.

Deleting Single Dictionary Elements

student={'Name': 'Dave', 'Age':17, 'Class': 'Software Development'}

>>>del student['Name']

This will only delete the Name record, including all information in the value.

Clearing Dictionaries

To clear all records in the dictionary:

>>>student.clear()

This will remove all records, but keep an empty 'student' dictionary.

Deleting Dictionaries

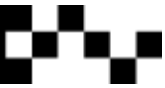
To delete the entire dictionary:

>>> del(student)

Calling specific details

>>> bob['name'], sue['pay']

What does Python display?



Splitting Dictionary Records

>>> bob['name'].split() [-1]

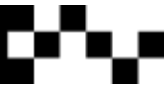
What does Python print?

Manipulating Dictionary Records

>>>sue['pay'] *= 1.10

>>> sue['pay']

What is Sue's pay total after this?



What are Tuples?

Tuples are a sequence of unchangeable objects. They are similar to lists, but the records within them cannot be changed or manipulated. Tuples use parenthesis () like lists use square brackets [] .

Example:

```
>>>tup1 = ('English', 'Maths',  2012, 2013)
>>>tup2 = (1, 2, 3, 4, 5, 6, 7)
>>>tup3 = ('a', 'b', 'c', 'd', 'e')
```

Accessing Tuple Values

It is similar to accessing values in lists.

print('Show the first word in the tup1: ' tup1[0])

What is the code for printing the fourth record in tup2?

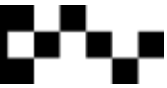
Updating Tuples

You cannot change values inside tuple elements. But we are able to take portions of an existing tuple to create new tuples.

Example:

```
>>>tup1 = (12, 34.56)
>>>tup2 = ('abc', 'xyz')
>>>tup3 = tup1 + tup2
>>>print(tup3)
```

What is displayed when you print tup3?



Deleting Tuples

Removing individual tuple elements is not possible. There is, of course, nothing wrong with putting together another tuple with the undesired elements discarded.

To explicitly remove an entire tuple, just use the del statement:

```
>>>tup = ('English', 'Maths', 2010, 2011)
>>>print (tup)
>>>del (tup)
>>>print ('After deleting tup : ')
>>>print (tup)
```

Fun with Tuples

```
>>> tuple2 = [‘Hi!’]

>>> print(tuple2 * 4)
```

What does this do?

Create your own tuple and get it to repeat 6 times.

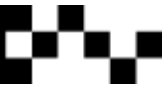
Tuple Concatenation

Concatenations is a fancy word for sticking something on the end of something else.

Tuple concatenation does exactly this.

```
>>>tupple3 = (1,2,3)
>>>tuple4 = (4,5,6)
>>>print(tuple3 + tuple4)
```

What displays on screen ?



Booleans and Tuples

You can check whether a particular integer or string is in a tuple.

numbers=(1,2,3,4,5,6,7,8,9)

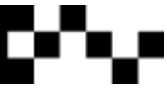
>>> print (3 in numbers)

What is the result?

>>> print (34 in numbers)

What is the result?

What happens when you check for a string or character in the *numbers* list?



Task 3: timetable.py

Input your current timetable into Python. Get it to display three separate lists, day, period and subject. Make sure it prints out similar to the timetable below.

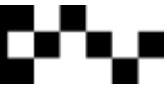
Monday
1 Maths
2 English
3 Science
4 Science
5 Info Tech
6 History

Tuesday
1 History
2 History
3 Science
4 English
5 Maths
6 Geography

Wednesday
1 Science
2 English
3 Info Tech
4 Geography
5 Geography
6 History

Thursday
1 English
2 Science
3 Geography
4 History
5 Info Tech
6 Info Tech

Friday
1 English
2 Science
3 Maths
4 Maths
5 Info Tech
6 History



Loops

Loops are an important part of programming, they are used to repeat an action over and over. There are two types of loops, *while* and *for*, both are used for separate purposes.

While Loops

While loops provide a way to code general loops, it repeatedly executes a block of indented statements as long as the test at the top keeps evaluating to a true value. It is called a "loop" because it keeps looping back to the start of the statement until the test becomes false.

```
>>> b = 1              b is equal to 1
>>>while b <=10:      while b is less than or equal to 10
        print(b)       print b
        b += 1         add 1 to b
```

The computer will run through the loop until it gets to 10, then it stops.

Infinite While Loops

If while loops are not executed correctly, they can become an infinite loop. Infinite loops are ones that never end.

To stop an infinite while loop, you type *break*.

```
while 1:
        name = input('Enter name: ')
        if name == 'quit':
                break;
```

What does this script do?

```
a = 0 ; b = 10
while a < b:
print(a)
a += 1
```

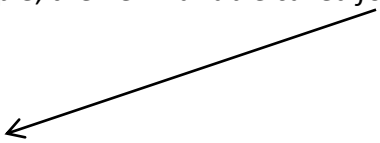
For Loops

For loops step through items in any ordered sequence. A *for* statement works on strings, lists, tuples and many other built-in objects.

```
>>> grocery=[‘bread’, ‘milk’, ‘cake’, ‘bikkies’, ‘custard’, ‘cereal’]

>>>for food in grocery:
        print(‘I want ‘ + food)
```

For statements loop through each item in the list and assign the element to a variable name.

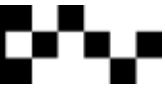
So, in the example, the new variable called *food* stores each item in the list called *grocery*, separately.



```
>>>for food in grocery:
        print(‘I want ‘ + food)

>>> grocery=[‘bread’, ‘milk’, ‘cake’, ‘bikkies’, ‘custard’, ‘cereal’]
```

It reads it as: food = bread
food = milk
food = cake …. and so on until the list is complete

Then displays the print statement for each item in the list.



Counting from 1 to 5

By using the **range()** function, we are able to use for loops to count.

```
>>>for number in range (1, 6):
        print (number)
```

The above code counts from 1 to 5. When declaring the range, always add 1 onto the last number.

```
>>>for number in range (20,71)

        print(number)
```

What does IDLE print?

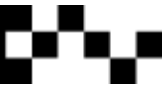
Counting Backwards From 5 to 1

```
>>>for number in range (5, 0, -1):
        print (number)
```

The numbers count backwards due to the -1 parameter. Range(5,0,-1) means start at 5, work backwards 1 place at a time until we get to zero.

```
>>>for number in range(20, 0, -2)

        print(number)
```

What does IDLE print?



Task 4: forLoop.py

Create a list that has six different animals in it – turkey, cow, horse, pig, dog and cat. Using a for loop, get Python to print the following statement: “A [animal] is the best”

Once you get the above task working, extend your script to have an IF statement. Look at the pseudocode below and think carefully about how you will achieve the following:

IF [animal] is a turkey, print “Gobble gobble”

ELSE IF [animal] is a cow, print “Moo Moo”

ELSE IF [animal] is a horse, print “Neigh”

ELSE IF [animal] is a pig, print “Oink oink”

ELSE IF [animal] is a dog, print “Woof Woof”

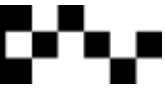
ELSE IF [animal] is a cat, print “Nom Nom”

Create integer variables to use as counters, one for the pets and another for the farm animals in the list.

For each animal in the IF statements, use the appropriate variable to add 1 onto the counter.

Add some zoo animals into the list and include a new *zoo* variable.

Print the results of all counters at the end.



Task 5: whileLoop.py

Turn the following pseudocode into python script.

temperature ← 115
WHILE temperature is greater than or equal to 80:
    PRINT temperature
    temperature – 1
END WHILE
PRINT 'The soup is cool enough'

EXTENSION : While, for loops and tuples

```
shop = list()                                    #creates an empty tuple to store the 'items' in
items = input('Press y if you want to enter more items: ')    # first message to print
while items == 'y':                              #tests that while items is equal to 'y'
   line = input('Next item: ')                   #ask for the next item if user entered 'y'
   shop.append(line)                             #appends(joins) the tuple together
   items = input('Press y if you want to enter more items:')  # ask user if they want another input
print ('Your shopping list is:')                 #if user types anything other than 'y' this is printed
for line in shop:                                #FOR LOOP to run through shop tuple
   print(line)                                   #print the line variable
```

Use the code above to change your *whileLoop.py* to include the feature of typing in temperatures of different foods and beverages.

Functions

Functions are chunks of executable code that can easily be used more than once in your program. Functions are used to maximise *code reuse* and minimise *code redundancy*.

The first thing you have to do is to define a function. But before you start calling and using your function you have to create one, to do this you type:

The **def** part of the function is short for **define**.

```
def functionName(x):
```

This creates the function

This names it

This is a variable (aka parameter) to be used within it (optional)

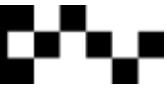
Not all functions do the same thing. They can create, store and return information. It is what you get them to do that makes them different.

A basic function looks like:

```
def myFunction():
    print('This function doesn\'t do anything.')
```

Defines the function

Body of the function



Functions with Parameters

```
def times(x, y):
    return (x *y)
```

This function will return x multiplied by y, and then wait for the user to fill in variables x and y.

```
def times(x, y):
    return (x *y)

>>> times(2,4)
```

By typing in the function name, then the numbers you want multiplied together, Python will automatically replace the x with 2 and y with 4.

Functions as a variable

```
>>>b = times(3.14, 4)
>>>b
```

This creates a new variable called b that stores the times function.

Functions and Strings

```
>>> times('Blah', 4)
'BlahBlahBlahBlah'
```

This stores Blah into the x variable, and 4 into the y variable. Therefore it prints Blah four times, one after another.

Plus Ten Function

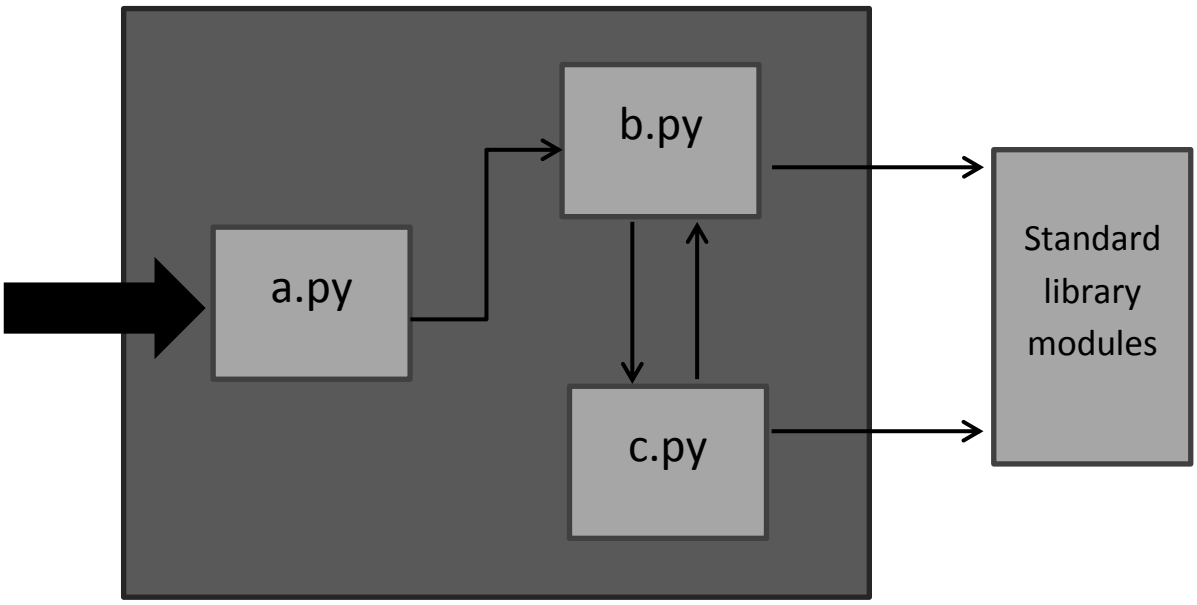
```
>>>def plusten(y)
        return(y+10)
>>> print(plusten(44))
```

What does this function do?

Modules

Modules are the highest-level program organisation unit. It packages code and data for reuse, similar to functions, but in a unique way. They provide an easy way to organise components into a system by serving as self-contained packages of variables known as *namespaces*.

In simple terms, every file of Python source code whose name ends in a *.py* extension is a module. Other files can access the items a module defines by importing that module; *import* operations essentially load another file, and grant access to that file's contents. The contents of a module are made available to the outside world through its attributes.



The image above sketches the structure of a Python program composed of three files: *a.py*, *b.py*, and *c.py*. The file *a.py* is chosen to be the top-level file; it will be a simple text file of statements, which is executed from top to bottom when launched. The files *b.py* and *c.py* are modules; they are simple text files of statements as well, but they are not usually launched directly. Instead, they are imported by other files that wish to use the tools they define.

How to make modules

Open a new Python file, save it as *goldfish.py*.

Type the following into it:

title = 'Most goldfish are orange.'

Save (don't forget the *.py* after the name)

Press F5 to run the script. Nothing will load.

Importing Modules

Python allows you to use custom modules. To import them into IDLE type:

import ModuleName

To put in action:

```
>>>import goldfish          #Run file; load the module as a whole
>>>print(goldfish.title)    #Use '.' to qualify which statement you want printed
```

What does IDLE print?

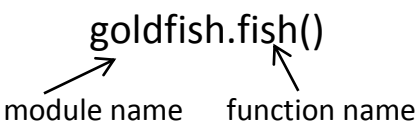
Create three new variables in the *goldfish.py* file and get them to print.

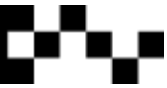
Functions in Modules

You can call functions within modules after you have imported them.

```
>>>def fish()
        print('Fish live in water.')
```

In IDLE, type:





About Modules

Importing modules is a delicate business. You can only import modules once each time you run IDLE. Although you can keep typing *import ModuleName* into IDLE, it won't import the same module twice. If you make changes to your module, you must reload IDLE before you can import the changes.

random()

What is random?

Random is something that it not predictable or based on chance, like rolling a dice or flipping a coin. Python has a built in module that you can import to give your programs the functionality of random.

To use random in Python, first you have to import a module called random.

```
import random      #type this at the very top of your script
```

Then to use it in your program you can:

```
variable = random.choice(array)
```

variable name to store random information

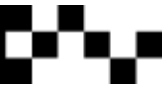
random module called "choice"

The array you want the random item to be taken from

Example:

```
import random
flavours = ['vanilla', 'triple chocolate', 'mint chock chip', 'rum and raisin', 'blueberry fudge']

picked = random.choice(flavours)
print (picked)
```

What does the above code do?



Task 6: ralf.py

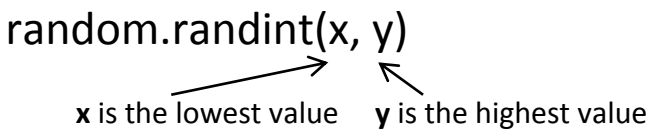
Ralf has a gemstone washing machine. He has 40 dirty rocks. Inside each rock is a gem. Some rocks don't have a gem, they will turn into mud, but Ralf will only know once the rock has been cleaned. He wants to know how many opals, rubies, turquoise, emeralds and diamonds there are in his bucket.

Write the pseudocode first.

Hint: Use random.choice()

random.int()

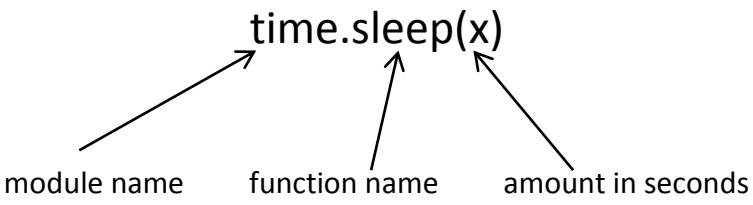
random.int() is a function within the random module that calls a random integer.



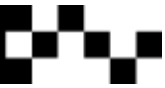
Like random.choice() you can store the random.int() in a variable, so it is easier to call upon the chosen integer throughout your program.

time()

time() is another built-in function that provides the ability to pause print statements. To use it, you must first import *time*.



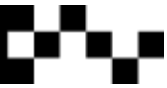
You don't need to store the time in a variable.



Task 7: dragon.py

Re-write this pseudocode into Python. Don't forget to write comments into your code.

```
import random, time

FUNCTION displayIntro
        PRINT 'You are in a land full of dragons. In front of you, you see two caves. In one
        cave, the dragon is friendly and will share his treasure with you. The other dragon is
        greedy and hungry, and will eat you on sight.'
END displayIntro

FUNCTION chooseCave
        cave ← empty string
        WHILE cave <> '1' and cave <> '2'                # <> means not equal to
                PRINT 'Which cave will you go into? (1 or 2)'
                cave ← input
        END WHILE
        return cave
END chooseCave

FUNCTION checkCave(chosenCave):
        PRINT 'You approach the cave…'
        Sleep for 2 seconds
        PRINT 'It is dark and spooky…'
        Sleep for 2 seconds
        PRINT 'A large dragon jumps in front of you! He opens his jaws and…'
        Sleep for 2 seconds

        friendlyCave ← random.randint(1, 2)

        IF chosenCave is equal to str(friendlyCave)
                PRINT 'Gives you his treasure!'
        ELSE
                PRINT 'Gobbles you down in one bite!"
        END IF
END checkCave

playAgain ← yes
WHILE playAgain is equal to yes or playAgain is equal to y
        displayIntro()
        caveNumber ← chooseCave()
        checkCave(caveNumber)

        PRINT 'Do you want to play again? (yes or no)'
        playAgain ← input
END WHILE
```



str.maketrans()

str.maketrans adds a little bit of fun into your Python program, it can encrypt and decrypt messages.

```
word = input('Type in a sentance: ')

table = str.maketrans(
   "cdefghijklmnopqrstuvwxyzab" , "abcdefghijklmnopqrstuvwxyz"
)

encrypt = word.translate(table)

print(encrypt)
```

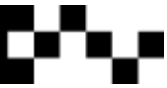
Let's go through it line by line:

word = input('Type in a sentence: ')
Stores the user input in a variable called *original*.

table = str.maketrans(
"cdefghijklmnopqrstuvwxyzab" , "abcdefghijklmnopqrstuvwxyz"
)
This calls maketrans and stores it in a variable called *table*. The first string is what the input will be converted to, the second string is the *original* alphabet.

encrypt = word.translate(table)
This creates a new variable called *encrypt*, which stores the information input into the *word* variable, then runs it through the translation variable called *table*.

print(encrypt)
This prints the encrypt variable, displaying the encrypted message.



Task 8: translate.py

Using str.maketrans translate this message:

xqg eio nqfzy ioewwx poww aq yoa acfd uei pfac xqgi tiqyievvfzy. yqg cejo woeizon e wqa dfzko xqg hoyez oeiwfoi acfd xoei. xqgi drfwwd pfww nojowqt ugiacoi ed xqg daeia aq pifao xqgi qpz tiqyievd. kqzyieagweafqzd qz ojoixacfzy xqg cejo ekcfojon, xqg cejo aq ho joix tiqgn qu xqgidowu.

With this encryption:

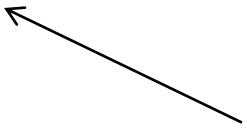
tzhsaiubrvcqjdewokxpfmlygn

What does it say?

Modulus %

Modulus finds the remainder.

```
>>>12%9
3
```



You can read the above as: 9, 10, 11, 12 – there are **three** numbers left over after 9.

Modulus isn't only to find the remainder, it is used to input data into your program.

Modulus d (%d)

%d reads decimal integers.

```
>>>camels = 42
>>>print('%d camels' % camels)
```

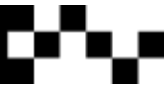
What does this code do?

```
>>>print ('%d %d %d' % (1, 2, 3))
```

What does this code do?

```
def times(number):
        for chosenNumber in range(1, 13):
                print ('%d x %d = %d' % (chosenNumber, number, chosenNumber*number))

times(5)
```

What does this code do?



Modulus s (%s)

%s reads strings.

```
>>>print('%s %s %s' % ('dogs', 'cats', 'camels'))
```

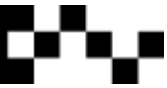
What does this code do?

```
>>>rabbits = 'bunnikins'
>>>print('%s are rabbits' % rabbits)
```

What does this code print?

```
>>>orange= 'goldfish'
>>>brown='horse'
>>>green='frog'
>>>print('%s %s %s', (orange, brown, green))
```

What does this code print?



Modulus g (%g)

%g reads floating point numbers.

>>>num = 56.383

>>>print('The number is: %g' % num)

What does this code do?

>>>print('Numbers %g and %g are floating points.' % (238.483, 5995.990)

What does this code do?

Putting them all together

>>> print('In %d years I have seen %g %s.' % (3, 5.75, 'dogs'))

Using modulus d, s and g write some code that reads variables that stores appropriate data. Write your code below:

Skill Building

Remember to write it in pseudocode first!

Skill 1: TrafficControl.py

Write a program that Police can use to check the speed of drivers. The Police input the speed zone before the speed checks take place. As cars go past, the Police get the speed the car was going, and displays a message saying whether they were going too fast, the correct speed or slow. The program should issue a fine if the car is going too fast: 5km over the limit: $120 fine, 10km over the limit: $230 fine, 20km over the limit: $450 fine, anything above 20 km the Police issue an Immediate Cancellation of Licence.

Skill 2: Airport.py

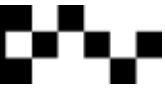
A plane with 300 passengers has landed. Write a program that sorts the passengers out. How many passengers have Australian passports and how many have International passports. You should be able to tell how many passengers have filled in a Customs Declaration Form and declared something to go through customs, and how many illegal items were seized and what these illegal items were, e.g. a crate of chickens, dried figs, wooden items with spiders, box of snakes, Asian fruits with bugs, beef jerky, drugs, fur coat with teeth and a moose head.

Skill 3: fishing.py

The ASDF (Australia's Special Department of Fishing) is checking the catches of recreational fishermen. They are looking for the amount and sizes of particular fish:

Fish Type	Minimum Size	Limit
Shark	50cm	3
Whiting	25cm	30
Sea Bass	1 meter	15
Flathead	25cm	20
Bream	20cm	23
Orange Roughie	30cm	12

If the ASDF finds fish that are below the legal minimum size, the fisherman is issued a $230 fine per fish. If the fisherman has fish above the minimum size they can keep it as long as they have a valid fishing license. If fishermen are caught with Abalone they are arrested and go to gaol. Boats have to be registered, sea worthy (e.g. have life jackets and a flair) and have a GPS, if they do not have any or all of these items they are issued with a warning and a $678 fine.



Skill 4: election.py

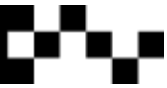
It's election time in Roughead Region! Fred Ferris and Georgia Green are both hoping to win the election, but it's up to the people to decide. Within the Region are six electorates, each with a different population:

Electorate	Population
Harrison	1023
Nimbottom	34,870
Goosehead	23,452
Davis	530
Ferdinand	67,001
Charles	10,080

Calculate the number of people who will vote for each candidate, and ultimately who is the winner of the election. Keep in mind there are children below 16, and elderly above 80 do not have to vote.

Skill 5: landcare.py

Landcare has three weeks to revegetate Erosion Point. They need 80 volunteers to revegetate 100 acres. Each volunteer can plant 30 trees a day. Landcare is going to use two types of tree; Spotted Gum and Stringy Bark. How many trees and what type do they need?



Input and Output

By this stage you would have become quite familiar with the print() function. Another, more controlled way to display data on the screen is to use an output file. Output files are basic text documents either created in Notepad or IDLE that read and interprets the data or strings within the file.

The built-in (standard) string module has some extremely useful operations that allows us to easily format output.

str() and str.format()

We've had a look at str() with the text based calculator on page 14.

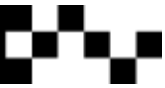
str() changes integers into strings so they can be printed as words.

```
>>>count = 23
>>>print(count)
```

What displays in IDLE?

```
>>>count = 23
>>> print(str(count))
```

What prints now?



str.format()

The str.format() method works like this:

>>>print('We are at {} and enjoy {}').format('school', 'Software Development'))

What does IDLE print?

The brackets and characters within them (called format fields) are replaced with the objects passed into the .format() method.

A number in the brackets can be used to refer to the position of the object passed into the .format() method, similar to an array.

>>> print('{0} is {1}'. format('programming', 'fun'))

>>> print ('{1} is {0}'. format('programming', 'fun'))

What does IDLE print?

You can use keywords in the format() method:

>>>print('I like {fruit} and {vegetable}.'.format(fruit= 'apples', vegetable='potato'))

What does IDLE print?

Escape Characters

When you get Python to print a lot of text, sometimes you want to format it a bit, escape characters let you do exactly that. An escape character is basically a \ (back slash), sometimes with a letter after it to explain to the computer what is supposed to happen.

Type	What it does	Example	Result
\n	Line feed (or an ENTER)	print('1\n2')	1 2
\'	To type a single quote	print('Don\'t do that!')	Don't do that!
\"	To type a double quote	print('He said \"Yes\"')	He said "Yes!"

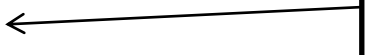
>>>print('Today is sunny. \n Yesterday it rained. \nTomorrow it\'s going to snow.')

When using the \n escape character be careful not to put a space between the \n and the next word, because Python will read the space and push the words across by one character on the next line.

Example:

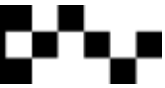
>>>print('Software Development\n English\n French ')

Software Development
 English
 French



You can see how the words on the lower lines have been pushed over because of the spaces.

Try the escape characters out and see what you can do!



Fancier Output Formatting

As mentioned before the str() function returns representations of data which are human-readable, while the repr() function is meant to generate representations which can be read by IDLE.

Example:

>>> s = 'Software Development'

>>>str(s)

What does IDLE PRINT?

>>>repr(s)

What does IDLE print?

>>>x = 4 *2

>>>y = 2 * 10

>>>s = 'The value of x is '+repr(x) + ', and y is ' + repr(y)

>>>print (s)

What does IDLE print?

Printing a Table with Columns and Rows

.rjust() justifies the text to the right.
.ljust() moves the text to the left
.center() moves the text to the centre.

The number in the parenthesis is the maximum allowable screen space.

```
>>> for x in range(1,11):
        print(repr(x).rjust(2), repr(x*x).rjust(3), end=' ')
        print(repr(x*x*x).rjust(4))
```

What does IDLE print?

The {} designate the position in the array, and the distance each column is away from each other, separated by a ' : '.

```
>>>for x in range(1,11):
        print('{0:2d} {1:3d} {2:4d}'.format(x, x*x, x*x*x))
```

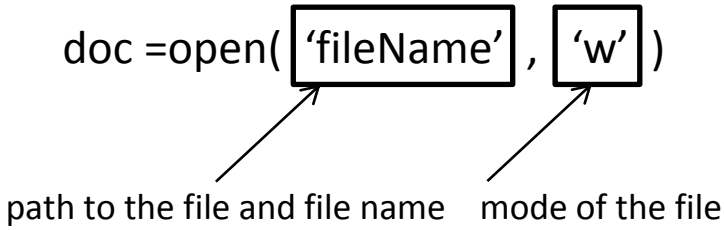
What does IDLE print?

Writing to External Text Files

open()

open() reads a file, it is written with the name of the file and the mode of the file.

Example:



The mode describes the way in which the file will be used. The mode can be:

- 'r' when the file can only be read
- 'w' when the file can be written to
- 'a' opens the file for appending. Meaning any data written to the file is automatically added to the end.

The mode is optional, but it is automatically assumed to be 'r' if it has not been included at the end.

Files are usually opened in text mode, meaning that you can read and write strings from and to a file. Let's take a look.

Open Notepad and create a text file called: '**test.txt**' - this is the file we are going to be working with. Close the text file.

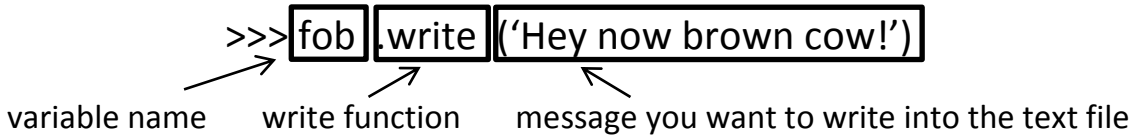
The first thing we have to do when working with text files is to create *file object* which is going to allow you to perform actions on the file. To do this we create a variable in IDLE.

```
>>> fob = open('C:/Documents/Python/test.txt', 'w')
```

Now we can use IDLE to write data to **test.txt**.

Every time you use the name of the variable "**fob**" you care calling the file path.

To write data to the text file you use:



close()

You must always close the file so that Python knows that you have finished writing to it.

>>>fob.close()

Now go and open the text file.

Reading from Text Files

Reading from text files does the opposite of writing to them.

>>>fob2=open('C:/Documents/Python/test.txt', 'r')

Use r instead of w

>>>fob2.read(3)

Read reads the bytes, 1 byte is usually a character or a number, so in the code above IDLE will print the first three letters.

To read the remainder of the file:

>>>fob2.read()

The lack of parameters in the parenthesis means that IDLE will read the remaining bytes in the file.

IDLE will print ' now brown cow', because it has already read the first three bytes.

Reading Line by Line

Create a variable to store the file object:

>>>fob3=open('C:/Documents/Python/test.txt', 'r')

>>>print(fob3.readline())

This reads the first line of the file up to the first line break.

>>>print(fob3.readlines())

Beware of the 's' on the end! **Readlines** does something different to **readline**.

Readlines reads the remaining text in the text file and displays it as a list.

Close all files: >>>fob.close()

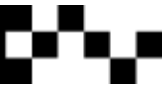
Writing Lines to a Text File

>>> fob = open('C:/Documents/Python/test.txt', 'w')

>>>fob.write('this is a new line \n horses are usually brown\n goldfish are orange\n rabbits are fluffy')

This creates four new lines in your **test.txt**.

>>>fob.close()



Modify Certain Lines

Restart IDLE, then create a new file object:

>>>fob=open('C:/Documents/Python/test.txt', 'r') #set the mode to 'r' (read)

Create a variable to store the 'readline**s**' list.

>>>readlist=fob.readlines()

>>>readlist

What does IDLE print?

>>>fob.close()

To modify a certain line in the list now you can call the list item, remembering that the first list item is [0].

>>>readlist[2]= "Elephants are grey"

>>>readlist

What does IDLE print?

The change is not saved to the text document yet. We have to create a new file object to do this.

>>>fob=open('C:/Documents/Python/test.txt', '**w**')

>>>fob.writelines(readlist)

>>>fob.close()

This writes the new data into the text file.

>>>print(readlist)

What does IDLE print?

Advanced Python

2D arrays

In math you would have been introduced to matrices. 2D arrays and matrices are the same thing; they just look a little different.

Example:

$$\begin{pmatrix} 3 & 5 & 7 \\ 4 & 0 & 2 \\ 1 & 2 & 5 \end{pmatrix}$$

Maths matrix

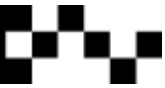
```
m =  [[3, 5, 7],
      [4, 0, 2],
      [1, 2, 5]]
```

Python matrix

Matrices in Python work the same as in maths.

$$\begin{pmatrix} 3 & 5 & 7 \\ 4 & 0 & 2 \\ 1 & 2 & 5 \end{pmatrix} + \begin{pmatrix} 1 & 2 & 3 \\ 3 & 1 & 2 \\ 2 & 3 & 1 \end{pmatrix} = \begin{pmatrix} 4 & 7 & 10 \\ 7 & 1 & 4 \\ 3 & 5 & 6 \end{pmatrix}$$

```
m =  [[3, 5, 7],
      [4, 0, 2],
      [1, 2, 5]]

m2 = [[1, 2, 3],
      [3, 1, 2],
      [2, 3, 6]]
```



The only problem with working with 2D arrays, is that they have to be the same size if you want to perform any mathematical function on them.