Python

Python Programming Language

Python is a **high-level**, general-purpose **programming language**. Its design philosophy emphasizes code **readability**.

It is an interpreted language as opposed to a compiled language, like C and C++. This makes it very good for prototyping solutions.

Assignments

x = 5

This instruction stores the value on the right-hand side in a piece of memory that can be referenced by the name on the left-hand side.

Variable Name Conventions

- **A variable name must start with a letter or the underscore character**. A variable name cannot start with a number. A variable name can only contain alpha-numeric characters and underscores (A-z, 0-9, and _) Variable names are case-sensitive (age, Age and AGE are three different variables).
- The convention for variable names is lowercase with words separated by underscores.
- Constants are not supported – convention is to give the names in UPPERCASE.

Print

print

- takes a list of elements in parentheses separated by commas
- if the element is in inverted-commas then it is printed as is
- if the element is a variable name outside inverted-commas it will print the value associated with the variable
- after printing moves on to a new line of output
- To move onto a newline in a print statement use '\n'

```
x = 12
print('My variable, x, has a value of: ', x, '\nThanks!')
```

Input from keyboard

```
X = input('Enter a value')
```

prints “Enter a value” on the python screen and waits until the user types something (anything), ending with Enter

The input is returned as a string (sequence of characters), no matter what is given, even a number ('1' is not the same as 1, different types)

We can force a type on the input e.g. make it an integer:

```
X = int(input('Enter an integer value'))
```

Arithmetic operators in Python

+	Add
-	Subtract
*	Multiply
/	Divide

- For integers – modulo operator:%
  - We will be using this occasionally, acts like a remainder of integer division – e.g. z == 2

Boolean operators

==	Equal	x == y
!=	Not equal	x != y
>	Greater than	x > y
<	Less than	x < y
>=	Greater than or equal to	x >= y
<=	Less than or equal to	x <= y

If statement

Simple if	**if** Boolean condition: Instructions
If/else	**if** Boolean condition: instruction set 1 **else**: instruction set 2
If/elseif/else	**if** Boolean condition: instruction set 1 **elif** Boolean condition: instruction set 2 **elif** Boolean condition: instruction set 3 **else**: Instruction set 4

Example of else if:

```
result = 69
if result >= 70 and result <= 100:
    print('You get a first')
elif result >= 60 and result < 70:
    print('You get a 2.1')
elif result >=50 and result < 60:
    print('You get a a 2.2')
elif result >= 40 and result < 50:
    print('You get a pass')
else:
    print('The questions didn’t suit you…')
```

Note- the colon ‘:’ indicates the end of the condition/start of instruction block.

Indentation is Part of the Syntax of Python

Inside control structures (if, while loops, for loops etc.) indentation delineates the scope of the structure. The scope ends the first time the code moves back to the level of indentation of the key word of the structure (if, else, while, for etc.).

e.g.

```
total = 0
x = 1
counter = 0
while x != 0:
    x = int(input('please enter a positive integer and I will add it:'))
    counter = counter + 1
    if counter==5:
        print('you have entered ',counter, 'numbers')
    total = total + x
print('total = ', total)
```

This set of instructions are inside the while loop

This print is outside the while loop

Python only recognizes sets of instructions when they are indented the same distance (***standard is 4 spaces***)

You must be very careful with the indentation

While loop

while loop will repeat the instructions in the loop body while the Boolean is `True`

If the Boolean expression never changes from the value true during the course of the loop, the loop will continue forever.

While loop

```
while Boolean condition:
        instruction set
Instruction set after loop
```

It is often crucial to test the loop initialization and termination – typically this is a big part of solution design.

While loop

while loop will repeat the statements in it's scope while the Boolean is `True`
If the Boolean expression never changes during the course of the loop, the loop will continue forever.

Problem

- Ask the user to enter 10 integers, one at a time (integers between 0 and 100)
- Count the number of results 50 or over
- We do not need to remember the entered results, just count the number over 50.

Solution

```
print('please enter 10 results,one at a time, when prompted')
i = 0
count = 0
while i < 10:
    result = int(input("Please enter a result:"))
    if result >= 50:
        count = count + 1
    i = i+1
print(count, ' got 50 or over')
```

for loop

For loops are useful if you know how many times you need to run the loop before the loop starts.

For loop

```
for element in collection:
    instruction set
```

```
for i in range(1,10):
  print(i)

for c in 'Hello World!':
  print(c)


for el in [1,3,5,10]:
  print(el)
```

For loop

The range function generates a sequence of integers

The range function takes 3 arguments:

- the **start** of the range. Assumed to be 0 if not provided
- the **end** of the range, but not inclusive (up to but not including the number). Required
- the **step** of the range. Assumed to be 1 if not provided

If only one argument is provided, it is assumed to be the end value

```
for num in range(1,5):
    print(num)
```

range represents the sequence 1, 2, 3, 4

This for loop assigns each of the values in the sequence to num,one at a time, and

prints each number (one number per line)

Nested loop

It is not unusual to have loops inside loops (or, in general control structures nested in control struectures) e.g.

```
i = 1
while i <= 5:
   j = 1
   while j <= 10:
      print(j, end='')
      j = j + 1
   i = i + 1
   print()
```