14. User Input

In a Python command line application you can display information to the user using the `print()` function:

```
name = "Roger"
print(name)
```

We can also accept input from the user, using `input()` :

```
print('What is your age?')
age = input()
print('Your age is ' + age)
```

This approach gets input at runtime, meaning the program will stop execution and will wait until the user types something and presses the `enter` key.

You can also do more complex input processing and accept input at program invocation time, and we'll see how to do that later on.

This works for command line applications. Other kinds of applications will need a different way of accepting input.

15. Control Statements

What's interesting to do with booleans, and expressions that return a boolean in particular, is that we can make decisions and take different roads depending on their `True` or `False` value.

In Python we do so using the `if` statement:

```
condition = True

if condition == True:
    # do something
```

When the condition test resolves to `True`, like in the above case, its block gets executed.

What is a block? A block is that part that is indented one level (4 spaces usually) on the right:

```
condition = True

if condition == True:
    print("The condition")
    print("was true")
```

The block can be formed by a single line, or multiple lines as well, and it ends when you move back to the previous indentation level:

```
condition = True

if condition == True:
    print("The condition")
    print("was true")

print("Outside of the if")
```

In combination with `if` you can have an `else` block, that's executed if the condition test of `if` results to `False`:

```
condition = True

if condition == True:
    print("The condition")
    print("was True")
else:
    print("The condition")
    print("was False")
```

And you can have different linked `if` checks with `elif`, that's executed if the previous check was `False`:

```
condition = True
name = "Roger"

if condition == True:
    print("The condition")
    print("was True")
elif name == "Roger":
    print("Hello Roger")
else:
    print("The condition")
    print("was False")
```

The second block in this case is executed if `condition` is `False`, and the `name` variable value is "Roger".

In a `if` statement you can have just one `if` and `else` checks, but multiple series of `elif` checks:

```
condition = True
name = "Roger"

if condition == True:
    print("The condition")
    print("was True")
elif name == "Roger":
    print("Hello Roger")
elif name == "Syd":
    print("Hello Syd")
elif name == "Flavio":
    print("Hello Flavio")
else:
    print("The condition")
    print("was False")
```

`if` and `else` can also be used in an inline format, which lets us return a value or another based on a condition.

Example:

```
a = 2
result = 2 if a == 0 else 3
print(result) # 3
```

16. Lists

Lists are an essential Python data structure.

The allow you to group together multiple values and reference them all with a common name.

For example:

```
dogs = ["Roger", "Syd"]
```

A list can hold values of different types:

and you check with `id(age)` you will find that `age` points to a different memory location. The original value has not mutated, we switched to another value.

22. Loops

Loops are one essential part of programming.

In Python we have 2 kinds of loops: **while loops** and **for loops**.

22.1. `while` loop

`while` loops are defined using the `while` keyword, and they repeat their block until the condition is evaluated as `False`:

```
condition = True
while condition == True:
    print("The condition is True")
```

This is an **infinite loop**. It never ends.

Let's halt the loop right after the first iteration:

```
condition = True
while condition == True:
    print("The condition is True")
    condition = False

print("After the loop")
```

In this case, the first iteration is ran, as the condition test is evaluated to `True`, and at the second iteration the condition test evaluates to `False`, so the control goes to the next instruction, after the loop.

It's common to have a counter to stop the iteration after some number of cycles:

```
count = 0
while count < 10:
    print("The condition is True")
    count = count + 1

print("After the loop")
```

22.2. `for` loop

Using `for` loops we can tell Python to execute a block for a pre-determined amount of times, up front, and without the need of a separate variable and conditional to check its value.

For example we can iterate the items in a list:

```
items = [1, 2, 3, 4]
for item in items:
    print(item)
```

Or, you can iterate a specific amount of times using the `range()` function:

```
for item in range(04):
    print(item)
```

`range(4)` creates a sequence that starts from 0 and contains 4 items: `[0, 1, 2, 3]`.

To get the index, you should wrap the sequence into the `enumerate()` function:

```
items = [1, 2, 3, 4]
for index, item in enumerate(items):
    print(index, item)
```

22.3. Break and continue