Making Decisions

Boolean Data Type

A boolean variable is either `True` or `False`.

```
loggedin = True

administrator = False
```

Boolean Operators - Comparison

- `==` `42 == 42` (Equality - do not confuse with = !)
- `!=` `10 != 12` (Inequality)
- `<` `2 < 3` (Less than)
  `“apple” < “banana”`
- `>` `4 > 3` (Greater than)
  `“dog” > “cat”`
- `<=` `5 <= 5` (Less than or equal to)
- `>=` `6 >= 6` (Greater than or equal to)

Boolean Operators - Comparison

- `==` `42 == 42` (Equality - do not confuse with = !)
- `!=` `10 != 12` (Inequality)
- `<` `2 < 3` (Less than)
  `"apple" < "banana"`
- `>` `4 > 3` (Greater than)
  `"dog" > "cat"`
- `<=` `5 <= 5` (Less than or equal to)
- `>=` `6 >= 6` (Greater than or equal to)

Note how comparison on strings compares alphabetically!

`12 < 2 == False` but `"12" < "2" == True`

Boolean Operators - Logic

- `not` `not False`
- `and` `True and True`
- `or` `True and False`

Boolean Operators - Logic

not	
Input	Output
False	True
True	False

and		
Input 1	Input 2	Output
False	False	False
False	True	False
True	False	False
True	True	True

or		
Input 1	Input 2	Output
False	False	False
False	True	True
True	False	True
True	True	True

Boolean Expressions

Can construct boolean expressions using these operators, like with numeric and string expressions:

```
name == “Alex”
```

```
age >= 18 or loggedin
```

Exercise: Writing Boolean Expressions

Say we have the following variables describing an item of food:

`colour`	Colour of food	`colour = "red"`
`category`	Category of food	`category = "fruit"`
`age`	Days since purchase	`age = 2`

Write a boolean expression that is True for all food items that are **not vegetables**, are **either brown or yellow**, and are **less than a week old**; and False for everything else.

Write a program with these variables that outputs the value of your expression. Try different combinations of values for the variables to test it works.

Exercise: Writing Boolean Expressions

Write a boolean expression that is True for all food items that are **not vegetables**, are **either brown or yellow**, and are **less than a week old**; and False for everything else.

```
category != “vegetable” and (colour == “brown” or colour
== “yellow”) and age < 7
```

If Statements

`if` *`boolean-expression`* `:` *`statement`*

- The boolean expression is evaluated.
- If it is True, then the statement is run.
- If it is False, then the statement is not run.

```
if age < 18 : print(“You are not old enough!”)
```

Exercise: Secret Word

Write a program that asks the user for a secret word of your choice. If they enter the correct secret word, then they are shown a message.

Exercise: Secret Word

Write a program that asks the user for a secret word of your choice. If they enter the correct secret word, then they are shown a message.

```
word = input("Enter the secret word: ")
if word == "swordfish" : print("Correct!")
```

Code Blocks

We probably want to run more than one statement if the condition is successful!

We use *indentation* to denote blocks of code.

- Indented code after a statement ending with a colon forms a block.
- Everything indented to the same level is part of the block.
- The block ends once the indentation ends.
- You can use however much indentation you want, but the standard in Python 3 is four spaces. The important thing is to be consistent.

Block If Statement

```
word = input("Enter the secret word: ")
if word == "swordfish":
    print("Correct!")
    print("Well done for knowing the secret word.")
print("Goodbye")
```

The first two print statements are part of the block. The last one is not, and will run whatever happens!

We will usually write our if statements in this block form.

Else

```
if boolean-expression:
    ... do stuff ...
else:
    ... do stuff ...
```

The `else` block is run if the boolean expression is false.

i.e. Exactly one of the two blocks will be run.

Exercise: Secret Word, Again

Write a program that asks the user for a secret word of your choice.

- If they enter the correct secret word, then they are shown a message consisting of several lines.
- If they enter the wrong word, then they are shown a warning message.

Exercise: Secret Word, Again

```
word = input(“Enter the secret word: “)
if word == “swordfish”:
    print(“Correct!”)
    print(“Well done for knowing the secret word.”)
else:
    print(“That’s not the right word!”)
```

Elif

```
if boolean-expression:
    ... do stuff ...
elif boolean-expression:
    ... do other stuff ...
elif boolean-expression:
    ... do some other stuff ...
else:
    ... do some other other stuff ...
```

We can have as many `elif`s as we need. The `else` must come last. The interpreter will try each expression in turn until one of them is True.

Elif

```
shape = input(“Enter a shape: “)
if shape == “triangle”:
    print(“Has three sides.”)
elif shape == “square” or shape == “rectangle”:
    print(“Has four sides.”)
elif shape == “pentagon”:
    print(“Has five sides.”)
elif shape == “hexagon”:
    print(“Has six sides.”)
else:
    print(“I don’t know that shape!”)
```

Nested Blocks

We can *nest* blocks inside other blocks using multiple levels of indentation.

```
age = int(input("Enter your age: "))
if age >= 18:
    word = input("Enter the secret word: ")
    if word == "swordfish":
        print("Welcome!")
    else:
        print("That's not the secret word!")
else:
    print("You are not old enough!")
```

Exercise: Quiz Program

Based on what you have learnt about decision making, write a simple quiz program.

The program should ask the user some questions, and the user should input their responses.

The program should tell the user whether they answered correctly.

The program could also keep a score of how many questions were answered correctly, to output at the end.

Try various types of questions to make sure you fully understand how boolean expressions, `if`, `else`, and `elif` work!

Short Circuit Evaluation

Consider a boolean expression using logical `and`:

```
x and y
```

How would we expect this to be evaluated?

Short Circuit Evaluation

Consider a boolean expression using logical `and`:

```
x and y
```

How would we expect this to be evaluated?

- Evaluate the boolean expression `x`.
- Evaluate the boolean expression `y`.
- Finally, evaluate the boolean expression `x and y`.

But what if `x` is False?

Short Circuit Evaluation

```
x and y
```

If `x` is False, then the full expression *must* evaluated to False!

So the interpreter does not both evaluating `y` in this case, because it does not need to.

A similar thing happens for `x or y`: if `x` is True, then there is no need to evaluate `y`.

This is called *short circuit evaluation* - the latter expression has been *short-circuited*.

Short Circuit Evaluation - Example Usage

```
if a / b == 4:
    print(“something”)
```

Not safe! What if `b` is 0?

Short Circuit Evaluation - Example Usage

```
if a / b == 4:
    print(“something”)
```

Not safe! What if `b` is 0?

```
if b != 0:
    if a / b == 4:
        print(“something”)
```

This is safe, but can we avoid the nesting?

Short Circuit Evaluation - Example Usage

Use short circuit evaluation!

```
if b != 0 and a / b == 4:
    print(“something”)
```

The first expression can short-circuit the second, to avoid division by zero.

A useful feature to be aware of! But watch out for it cropping up where you don’t expect it.