cct College Dublin
Computing • IT • Business

PROGRAMMING: OBJECT-ORIENTED APPROACH 

OPERATORS AND CONDITIONALS

- Press `Space` to navigate through the slides
- Use `Shift+Space` to go back
- Save as **PDF**:
  - Open **Chrome** then **click here**
  - Press **Ctrl+P**/**Cmd+P** to print
    - Destination: **Save as PDF**
    - Layout: **Landscape**
  - Press **Save** button

OPERATORS

- Operators are symbols built into python that allow for special functionality such as addition, subtraction and validation.

ARITHMETIC OPERATORS

- Arithmetic operators are operators used to do basic arithmetic (hence the name).
- What this means is that they are the basis for any calculations you need to make in a program.
- Most of these will be pretty obvious so I will try and make it more interesting by telling you some of the usefulness (weirdness) you wouldn't expect out of them, as well as some use cases.

ADDITION

- This should be pretty basic and boring, in fact you have already seen it before, but for the sake of completeness, here is how to do addition in python:

```
print(5 + 3) # Prints: 8
```

- You can also use variables:

```
sum_value = 5 + 8
print(sum_value) # Prints 8
```

ADDITION

- You can also use variables in place of numbers:

```
sum_2 = 6 + sum_value
print(sum_2) # Prints: 14
```

- Well that was boring, **but**... there are more interesting things you can do with addition.

ADDITION

- For example, if you have multiple lists you can actually add them together to combine the two (order matters):

```
my_list = [1,2,3,4] # Initialize my_list

my_list_2 = [5,6,7,8] # Initialize my_list_2

my_list = my_list + my_list_2 # Take the current value of my_list and add my_list_

print(my_list) # Prints: [1, 2, 3, 4, 5, 6, 7, 8]
```

- As you can see in the above example, when the **`my_list_2`** variable is added to the first **`my_list`** variable it is tacked on to the end of it (called *concatenation* in computer science).

ADDITION

- Here is where things get weird, if you recall in the first challenge (yes you should do those), I mentioned that strings are **like** lists.
- This means operands also work on strings:

```
name = "Hello " # Set name variable to an string 'hello '

name = name + "World!" # Take the current name value and add 'world' to it

print(name) # Prints: Hello World!
```

SUBTRACTION

- Just like addition, subtraction is somewhat boring the syntax for it is pretty simple:

```
print(5 - 3) # Prints: 2
```

- You can also use variables

```
value = 5 - 8
print(value) # Prints -3
```

- You can also use variables in place of numbers

```
value_2 = value - 6
print(value_2) # Prints: -9
```

- Unlike addition there are no fun tricks with it though, subtracting from a string or list will just give you an error.

MULTIPLICATION

- Now multiplication is back to getting weird.
- First lets get the simple syntax down:

```
print(5 * 3) # Prints: 15
```

- You can also use variables:

```
value = 5 * 8
print(value) # Prints 40
```

- You can also use variables in place of numbers:

```
value_2 = value * 6
print(sum_2) # Prints: 120
```

MULTIPLICATION

- Now for the weird part, with lists and strings you can multiply them, and the results are very fun...

```
my_list = [1,2,3,4] # Initialize my_list

my_list = my_list * 3 # Take my_list and multiply it by 3

print(my_list) # Prints: [1,2,3,4,1,2,3,4,1,2,3,4]
```

- As you can see, when you multiply a list (or string) by a number, it *concatenates* the value of the list (or string) the number of times you multiply it by.
- So in this case since the list is multiplied by 3, the value of the list is present 3 times in a row.

DIVISION

- And going back to the boring we have division, division has no special uses, but it does have 2 forms **integer** and **floating point** division.

DIVISION

FLOATING POINT DIVISION

- This is your typical division, it will **always** return a float. The syntax is as follows:

```
print(5 / 2) # Prints: 2.5
```

- You can also use variables:

```
value = 5 / 3
print(value) # Prints 1.6666666666666667
```

- You can also use variables in place of numbers

```
value_2 = value / 2
print(value_2) # Prints: 0.8333333333333334
```

DIVISION

INTEGER DIVISION

- This sort of division will **always** return an int.
- If your value comes out to a float (anything with a decimal) then it *takes the floor* of the division (always rounds down even if above 0.5).
- The syntax for integer division is pretty simple:

```
print(5 // 3) # Prints: 1
```

DIVISION

INTEGER DIVISION

- You can also use variables:

```
value = 5 // 8
print(value) # Prints 0 (Remember it ALWAYS rounds down)
```

- You can also use variables in place of numbers

```
value_2 = value // 6

print(value_2) # Prints: 0
```

SHORTCUTS

- Many of the operations you are doing are going to involve taking the original value of a variable doing an operation and storing the result back in the variable.
- For example:

```
variable_1 = 5 # Initialize the variable to 5

variable_1 = variable_1 + 5 # Take the current value of the variable and add 5

variable_1 = variable_1 - 5 # Take the current value of the variable and subtract

variable_1 = variable_1 / 2 # Take the current value of the variable and floating

variable_1 = variable_1 // 2 # Take the current value of the variable and integer
```

SHORTCUTS

- For addition, subtraction, multiplication and division there is a shortcut to do the operations shown in the previous slide.
- The general form is `variable_name <operator>= value`

```
variable_1 = 5 # Initialize the variable to 5

variable_1 += 5 # Take the current value of the variable and add 5

variable_1 -= 5 # Take the current value of the variable and subtract 5

variable_1 /= 2 # Take the current value of the variable and floating point divide

variable_1 //= 2 # Take the current value of the variable and integer divide by 2
```

MODULUS

- What this actually does is returns the **remainder** to the division of the two terms.
- This is not commonly used other than to check if something is *evenly divisible* by another number.
- This is because if a number is divisible by another then the **modulus** will be 0:

```
print(5 % 3) # Prints: 2
```

MODULUS

- You can also use variables:

```
value = 10 % 5

print(value) # Prints 0 (Therefore 10 is evenly divisible by 5)
```

MODULUS

- You can also use variables in place of numbers:

```
value_2 = value % 6

print(value_2) # Prints: 0 (Because 0/anything is always 0)
```

- Note that this is an incredibly slow way to do this check, but it is used often enough that it's worth learning.

LOGICAL OPERATORS

- Logical operators are symbols that are used to make comparisons between values.
- They all return *Boolean* values when used (True or False), and are useful especially when combined with *if statements* (explained later).
- All of these comparisons can be made with int's or float's

LOGICAL OPERATORS

GREATER THAN

- Used to check if a value is larger than another value:

```
print(5 > 3) # Prints: True; since 5 is greater than 3

result = 5 > 3 # You can store the result in a variable

print(result) # Prints: True

print(3 > 5) # Prints: False; since 3 is NOT greater than 5

print(5 > 5) # Prints: False; since 5 is NOT greater than 5 (they are equal)
```

LOGICAL OPERATORS

GREATER THAN OR EQUAL TO

- Used to check if a value is larger than **or** equal to another value:

```
print(5 >= 3) # Prints: True; since 5 is greater than 3

result = 5 >= 3 # You can store the result in a variable

print(result) # Prints: True

print(3 >= 5) # Prints: False; since 3 is NOT greater than 5

print(5 >= 5) # Prints: True; since 5 is equal to 5
```

LOGICAL OPERATORS

LESS THAN

- Used to check if a value is smaller than another value:

```
print(5 < 3) # Prints: False; since 5 is NOT less than 3

result = 5 < 3 # You can store the result in a variable

print(result) # Prints: False

print(3 < 5) # Prints: True; since 3 is less than 5

print(5 < 5) # Prints: False; since 5 is NOT less than 5 (they are equal)
```

LOGICAL OPERATORS

LESS THAN OR EQUAL TO

- Used to check if a value is smaller than **or** equal to another value:

```
print(5 <= 3) # Prints: False; since 5 is NOT less than OR equal to 3

result = 5 <= 3 # You can store the result in a variable

print(result) # Prints: False

print(3 <= 5) # Prints: True; since 3 is less than 5

print(5 <= 5) # Prints: True; since 5 is equal to 5
```

LOGICAL OPERATORS

NOT

- Not is an operand that flips the Boolean value of what follows.
- So something that would be True will be False and something that is False would be True.
- The syntax is `not <boolean>` here is an example:

```
print(not 5 < 3) # Prints: True; since the statement 5 < 3 evaluates to False

result = not 5 < 3 # You can store the result in a variable

print(result) # Prints: True

print(not 3 < 5) # Prints: False; Since the statement 3 < 5 evaluates to True
```

LOGICAL OPERATORS

AND

- The and operand takes two Boolean values, if they are both True it will return True, otherwise it is always False.
- The syntax is `<boolean> and <boolean>` for example:

```
print(5 < 3 and 8 > 6) # Prints: True; since both statements evaluate to True

print(5 > 3 and 8 > 6) # Prints: False; since the first statement evaluates to Fal
```

- I have provided a table below which will tell you how this works in all situations, these tables are called *truth tables* and they are quite useful. Assume that *a* and *b* are placeholders for statements that evaluate to True or false.

a	b	a and b	Example
True	True	True	`5 < 3 and 8 > 6`
False	True	False	`8 > 6 and 5 > 3`
True	False	False	`5 > 3 and 8 > 6`
False	False	False	`5 < 3 and 8 > 6`

LOGICAL OPERATORS

OR

- The *or* operand takes two Boolean values, if **either** are True it will return True, if **both** are False it will return False.
- The syntax is `<boolean> or<boolean>` for example:

```
print(5 < 3 or 8 > 6) # Prints: True; since both statements evaluate to True

print(5 > 3 or 8 > 6) # Prints: True; since the second statement evaluates to True
```

- Here is the truth table for the or operator:

a	b	a or b	Example
True	True	True	`5 < 3 or 8 > 6`
False	True	True	`8 > 6 or 5 > 3`
True	False	True	`5 > 3 or 8 > 6`
False	False	False	`5 < 3 or 8 > 6`

LOGICAL OPERATORS

IN

- The *in* operator is as far as I know one that is specific to python (and really useful).
- It takes two operands one being a value (int, string etc.) and the other being a collection of some sort (list, string).
- The operator will return True if the value is in the collection, for example lets say you wanted to check if a name taken from the command line is john, the code would look like this:

```
name = input("Enter name: ") # Take someones name from the command line

print("john" in name) # Prints: True if john is name given, or False otherwise
```

LOGICAL OPERATORS

IN

- Another example would be checking if a list contains a specific int:

```
print(4 in [1,2,3]) # Prints: False since no 4 is present in the list

print(4 in [1,2,3,4]) # Prints: True since 4 is present in the list
```

LOGICAL OPERATORS

BITWISE OPERATORS

DISCLAIMER: If you are new to programming feel free to skip on to conditionals as this section won't be relevant for you.

- For anyone who has used lower level languages bitwise operators are available in python (binary AND, OR, XOR, Ones Compliment, Left and Right shifts). I am not going to go into detail, but if you are interested here is the syntax:

Operator	Description	Example
&	Binary AND: Operator copies a bit to the result if it exists in both operands	(a & b) (means 0000 1100)
|	Binary OR: It copies a bit if it exists in either operand.	(a | b) = 61 (means 0011 1101)
^	Binary XOR: It copies the bit if it is set in one operand but not both.	(a ^ b) = 49 (means 0011 0001)
!	Binary Ones Complement: It is unary and has the effect of 'flipping' bits.	(~a) = -61 (means 1100 0011 in 2's complement form due to a signed binary number.
<<	Binary Left Shift: The left operands value is moved left by the number of bits specified by the right operand.	a << 2 = 240 (means 1111 0000)
>>	Binary Right Shift: The left operands value is moved right by the number of bits specified by the right operand.	a >> 2 = 15 (means 0000 1111)

Chart taken from here

CONDITIONALS
IF, ELSE, AND ELIF STATEMENTS

- Conditional statements are a statement that takes a logical operator and executes code if the operator is True.
- Technically this is also a part of control flows so I will just show you one of what's to come in the next module.

CONDITIONALS

IF

- This conditional is exactly how it sounds, **if** some operator or boolean is True then **do something**. The syntax for this statement looks something like this:

```
if condition:
    # Do stuff at this indentation level
```

- You'll notice a few things about this:

1. First you need to put a colon after your if statement:
    - This is because you can combine statements together (I will show this later), so the colon is there to say "This if statement is complete".
2. After the colon you will need to *indent* all the code you want to run by **at least one space or tab** (Typical convention is 4 spaces or 1 tab):
    - This tells python which code you want to run if the condition is True, and which to run regardless of if the statement is True.

CONDITIONALS

IF

- For example: lets say you have a number (*x*) and **if** x < 3 you want to **add 2 to it**, and **no matter what** the value of x is you want to **print x** the code looks like this:

```
x = 2 # Setting up the x variable

if x < 3:
    x += 2 # This will run if x < 3, otherwise it will be skipped over

print(X) # Since this is on a lower indentation level, this code will run regardle

if x < 3:
    x += 2 # This will run if x < 3, otherwise it will be skipped over

print(X) # Since this is on a lower indentation level, this code will run regardle
```

- Try to figure out what happens without reading my explanation below, once you have a guess, keep reading.

CONDITIONALS
IF

What happens in the example:

1. Setting up a variable called x, which is set to 2
2. Because at this point x < 3 the code will add 2 to the current value of x, making it 4
3. Since the value of x is 4 the print statement will print 4
4. Since x is now **not** less than 3 nothing will be added to the value
5. Since the value of x is still 4 the print statement will print 4

CONDITIONALS

ELSE

- `else` statements are used in conjunction with if statements.
- They allow you to set what should happen if the if statements' condition is False.
- Here is the syntax:

```
if condition:
    # Do stuff
else:
    # Do different stuff (Only happens if the above if statement condition is Fals
```

CONDITIONALS
ELSE

- Modifying the earlier example, let's setup two sets of if-else statements in which if x < 3 we take the current x value and add 2 to it, otherwise (else) subtract 1.
- Here is the code:

```
x = 2 # Setting up the x variable

if x < 3:
    x += 2 # This will run if x < 3, otherwise it will be skipped over
else:
    x -= 1 # This will run if x is not less than 3

print(x) # Since this is on a lower indentation level, this code will run regardle

if x < 3:
    x += 2 # This will run if x < 3, otherwise it will be skipped over
else:
    x -= 1 # This will run if x is not less than 3

print(x) # Since this is on a lower indentation level, this code will run regardle
```

- Try to figure out what happens without reading my explanation below, once you have a guess, keep reading.

CONDITIONALS

ELSE

- What happens in the example:

1. Setting up a variable called x, which is set to 2
2. Because at this point x < 3 the code will add 2 to the current value of x, making it 4
3. Since the value of x is 4 the print statement will print 4
4. Since x is now **not** less than 3 the else statement is invoked and x has a 1 subtracted from it
5. Since the value of x is now 3 the print statement will print 3

source

CONDITIONALS
ELIF

- `elif` statements are a way to chain if statements.
- For example lets say you had 4 conditions you wanted to check for you could combine them together using logical operators, or to make it more legible you could have 4 different elif statements.
- The syntax is as follows:

```
if conidion: # You MUST have an initial if statement to use an elif statement
    # Do stuff

elif condition_2:
    # Do other stuff

else: # This is optional, but useful if you want a catch-all condition
    # Do other other stuff
```

CONDITIONALS

ELIF

- Let's take a more realistic example, let's say you wanted to make a program that takes in input from the command line and then if it is a value between 1 and 5, print the text version of the number (i.e. 5 would print "Five").
- Here is the code:

```
user_value = int(input("Enter a number between 0 and 5: "))

if user_value == 0:
    print("Zero")
elif user_value == 1:
    print("One")
elif user_value == 2:
    print("Two")
elif user_value == 3:
    print("Three")
elif user_value == 4:
    print("Four")
elif user_value == 5:
    print("Five")
else: # If value is not between 0 and 5
    print("Value provided is not between 0 and 5!")
```