

# Flow control: conditionals and loops

---

Look at conditionals and loops, called flow control

1. Concept of flow control
2. Logical variables
3. The `if` block
4. The `for` loop
5. The `while` loop

# Flow control

- Flow control allows you to
  - Selectively execute a block of code:  
the `if - elif - else` block
  - Repeatedly execute a block of code a fixed number of times:  
the `for` loop
  - Repeatedly execute a block of code until a criterion is reached:  
the `while` loop
  - Recover from errors:  
the `try - catch` block (not covered in this course)
- These are the main programming constructs for specifying what your program does
- The constructs are similar in all computing languages

# Logical expressions

- First, need a logical variable, that takes one of two values: **True** and **False**

```
a = True  
b = False
```

- Can also set values using logical expressions, e.g. using comparison operators:

```
x = 10.0      # create a float  
a = x > 5.0    # logical test (comparison) is True - value of a is True  
b = x > 11.0   # logical test (comparison) is False - value of b is False
```

- Can also use logical (boolean) operators **and**, **or**, **not**. Often used with comparison operators:

```
x = 10.0  
c = x > 5.0 and x > 11.0  # one logical test is True, one is False - value of c is False  
d = x > 5.0 or x > 11.0   # one logical test is True, one is False - value of d is True
```

# The `if` block

- The simplest is the single `if` block

```
x = 10.0

if x > 5.0:                                # Start of if block (note the :)
    print("x greater than 5.0")           # End of if block
    x += 3.0                              # Does NOT need to be blank
print("Value of x is " + str(x))
```

- Note the indentation - the code inside the `if` block MUST be indented. This is part of the language syntax.
- The execution continues after the `if` block

# The `if-else` block

- The next level of complexity is the `if-else` block

```
x = 10.0

if x >= 5.0:                                # if block
    print("x greater than or equal to 5.0")
    x += 3.0
else:                                        # else block
    print("x less than 5.0")
    x += 10.0

print("Value of x is " + str(x))
```

- Only *one* of the two blocks gets executed, then execution continues after the close of the second block.
- Remember that the indentation is absolutely critical.

# The `if-elif-else` block

- The most general format is the `if-elif-else` block

```
x = 10.0

if x > 5.0:                                # if block
    print("x greater than 5.0")
    x += 3
elif x > 7.0:                              # elif block
    print("x greater than 7.0")
    x += 4
else:                                       # else block
    print("x is small number")
    x += 10

print("Value of x is " + str(x))
```

- Note that the first `True` block is executed, then control jumps to the end of the `if-elif-else` block. So in this example the second block can *never* be executed.
- • Note also that we can have multiple `elif` blocks: `if – elif – elif - ... - else`

# The `for` loop

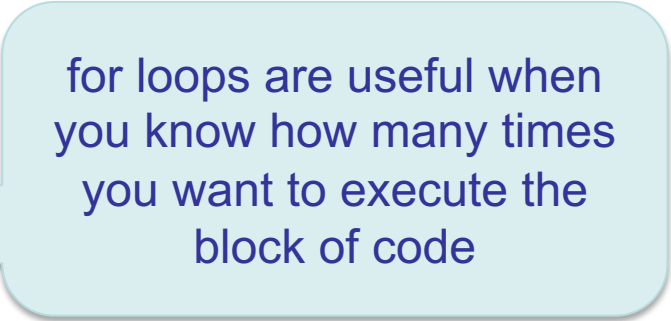
- Loops allow execution of a block of code multiple times. The simplest is the `for` loop:

```
import math

delta = 0.04

for i in range(100):
    t = i*delta
    print("t is " + str(t) + " cos(t) is " + str(math.cos(t)))

print("End of loop")
```



- Will go round the the loop executing the indented block with  $i = 0, 1, 2, \dots, 99$
- All parameters must be of type `int` and note that the loop stops “before” stop is reached
- Full syntax of range is:  
`range(start, stop, increment)`
- Must specify value of stop. If start and increment are not given, default values are used:
  - `start = 0, increment = 1`



# The `while` loop

- The `while` loop is a more flexible loop, that runs while a criterion is `True`

```
import math

x = 0.0
while x < 10.0:
    x += 1.0 + math.sqrt(x)
    print("value of x is " + str(x))

print("Final value of x is " + str(x))
```

while loops are useful when you know what the end point (stopping condition) is, but don't know how many times the loop has to be executed to reach this point

- This will go round the loop until the logical statement evaluates to `False`
- Note that this statement is tested *before* the loop is executed
- The `while` loop is a powerful loop, but remember that you must update the loop variable within the loop, or it will loop for ever - a very common bug!