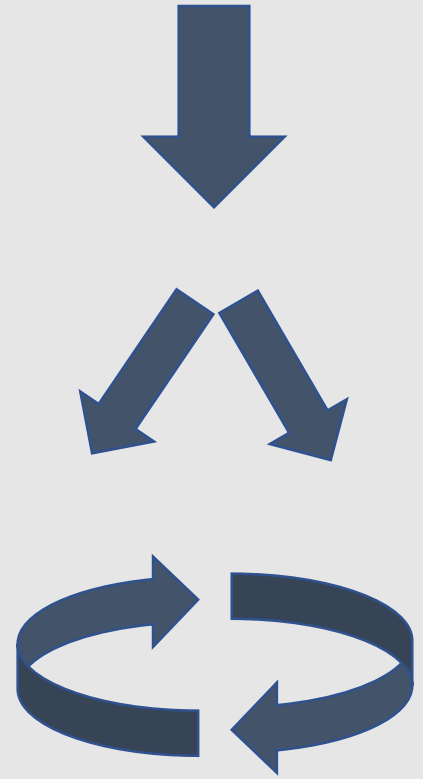
Python

Selection, Iteration, Operators, Functions

Sequence, Selection, Iteration

- **Sequence** mandates that statements be executed in order (line by line)
- **Selection** (conditional) statements will execute a **block of code once** when the condition is true
- **Iteration** allows us to **repeat** statements within a block **whilst** the condition is **true**



In this lecture

- Selection
- Operators
  - Comparison
  - Arithmetic
  - Logical
- Iteration
- Functions

Selection in Python

if

In[ ]:

```
1 | score = 45
2 | if score >= 40 :
3 |     print("You passed!")
```

In other words

```
score = 45
if score >= 40 :
```

Is the value of score (45) greater than (or equal) the value of 40?

Is 45 greater than 40?

True or False?

if

In[ ]:

```
1 | score = 45
2 | if score >= 40 :
3 |     print("You passed!")
```

if - running

In[ ]:
```
1 | score = 45
2 | if score >= 40 :
3 |     print("You passed!")
```

```
You passed!
```

if… else

In[ ]:

```
1 | score = 35
2 | if score >= 40 :
3 |     print("You passed!")
4 | else
5 |     print("Try again...")
```

In other words

```
score = 35
if score >= 40 :
```

Is 35 greater or less than 40?

In other words

```
score = 35
if score >= 40 :
```

Is 35 greater or less than 40? **LESS**

Therefore, if score is NOT greater than 40, can the conditional statement execute?

if… else

In[ ]:

```
1 | score = 35
2 | if score >= 40 :
3 |     print("You passed!")
4 | else :
5 |     print("Try again...")
```

if…else - running

In[ ]:
```
1 | score = 35
2 | if score >= 40 :
3 |     print("You passed!")
4 | else :
5 |     print("Try again...")
```

```
Try again...
```

elif (**el**se **if**)

In[ ]:

```
1 | score = 55
2 | if score >= 40 :
3 |     print("You passed!")
4 | elif score >= 50 :
5 |     print("You scored a C")
6 | else :
7 |     print("Try again...")
```

elif (**el**se **if**)

In[ ]:
```
1 | score = 55
2 | if score >= 40 :
3 |     print("You passed!")
4 | elif score >= 50 :
5 |     print("You scored a C")
6 | else :
7 |     print("Try again...")
```

```
You passed!
```

elif (**el**se **if**)

In[ ]:

```
1 | score = 45
2 | if score >= 50 :
3 |     print("You scored a C!")
4 | elif score >= 40 :
5 |     print("You passed!")
6 | else :
7 |     print("Try again...")
```

elif (**el**se **if**)

In[ ]:
```
1 | score = 45
2 | if score >= 50 :
3 |     print("You scored a C!")
4 | elif score >= 40 :
5 |     print("You passed!")
6 | else :
7 |     print("Try again...")
```

elif (**el**se **if**)

In[ ]:

```
1 | score = 45
2 | if score >= 50 :
3 |     print("You scored a C!")
4 | elif score >= 40 :
5 |     print("You passed!")
6 | else :
7 |     print("Try again...")
```

elif (**el**se **if**)

In[ ]:
```
1 | score = 45
2 | if score >= 50 :
3 |     print("You scored a C!")
4 | elif score >= 40 :
5 |     print("You passed!")
6 | else :
7 |     print("Try again...")
```

```
You passed!
```

Selection in C and Java

- Other languages: C, C++, C# and Java feature a switch statement and a ternary (?) operator as alternate ways to select between blocks of code.
- Switch is not present in Python, nor is the '?' used.
- But the ternary functionality can be performed with if and else in the same line (introduced in Python 2.5):

```
value_if_true if condition else value_if_false
```

Comparison Operators

Operators

Greater than	>
Less than	<
Greater than or equal	>=
Less than or equal	<=
Equality	==
Not equal	!=

Operators

Operator	Output
x == y	True if x and y have same value
x != y	True if x and y do not have same value
x < y	True if x is less than y
x > y	True if x is greater than y
x <= y	True if x is less than or equal to y
x >= y	True if x is greater than or equal to y

Operators

In[ ]:

```
1 | score = 45
2 | score >= 40
3 |
```

Operators

In[ ]:
```
1 | score = 45
2 | score >= 40
3 |
```

```
True
```

Operators

In[ ]:

```
1 | score = 45
2 | score == 45
3 |
```

Operators

In[ ]:
```
1 | score = 45
2 | score == 45
3 |
```

```
True
```

Operators

In[ ]:

```
1 | score = 45
2 | score == '45'
3 |
```

Operators

In[ ]:
```
1 | score = 45
2 | score == '45'
3 |
```

```
False
```

Operators

In[ ]:

```
1 | score = 45
2 | score != 45
3 |
```

Operators

In[ ]:
```
1 | score = 45
2 | score != 45
3 |
```

```
False
```

Arithmetic Operators

Operators

Addition	+
Subtraction	-
Division	/
Modulus	%
Multiplication	*
Floor Division	//
Power of	**

Operators

```
Operator     |  Output

 5 + 5       |  10
 5 - 4       |  1
 5 / 2       |  2.5
 50 % 4      |  2     (mod gives the remainder)
 5 * 10      |  50
 5 // 2      |  2     (round 2.5 down)
 5 ** 2      |  25    (5 x 5)
```

Operators

In[ ]:
```
1 | 8 ** 2 / 4
2 |
```

Operators

```
In[ ]: 1 | 8 ** 2 / 4
       2 |
```

Operators

In[ ]:
```
1 | 64 / 4
2 |
```

```
16
```

Operators

In[ ]:

```
1 | 4 + 4 ** 2
2 |
```

Operators

In[ ]:

```
1 | 4 + 4 ** 2
2 |
```

Operators

In[ ]:
```
1 | 4 + 16
2 |
```

```
20
```

Operators

In[ ]:

```
1 | (4 + 4) ** 2
2 |
```

Operators

```
In[ ]: 1 | (4 + 4) ** 2
       2 |
```

Operators

In[ ]:
```
1 | 8 ** 2
2 |
```

64

Logical Operators

Operators

Logical AND	and
Logical OR	or
Logical NOT	not

Operators

Operator	Output
x or y	Either x or y can be be True
x and y	Both x and y have to be True
not x	True only if x is False

Operators

In[ ]:

```
1 | score = 45
2 | if score >= 40 and score <= 100 :
3 |     print("You passed!")
```

Operators

In[ ]:

```
1 | score = 45
2 | if score >= 40 and score <= 100 :
3 |     print("You passed!")
```

```
You passed!
```

Operators

In[ ]:

```
1 | score = 101
2 | if score < 0 or score > 100 :
3 |     print("Incorrect mark!")
```

Operators

In[ ]:

```
1 | score = 101
2 | if score < 0 or score > 100 :
3 |     print("Incorrect mark!")
```

```
Incorrect mark
```

Operators

In[ ]:

```
1 | score = -1
2 | if score < 0 or score > 100 :
3 |     print("Incorrect mark!")
```

Operators

In[ ]:

```
1 | score = -1
2 | if score < 0 or score > 100 :
3 |     print("Incorrect mark!")
```

```
Incorrect mark
```

Iteration in Python

while

In[ ]:
```
1 | i = 0
2 | while i < 3 :
3 |     print("Loop", i)
4 |     i += 1
```

while - step

In[*]:
```
1 | i = 0
2 | while i < 3 :
3 |     print("Loop", i)
4 |     i += 1
```

while - step

In[*]:
```
1 | i = 0
2 | while i < 3 :
3 |     print("Loop", i)
4 |     i += 1
```

while - step

In[*]:
```
1 | i = 0
2 | while i < 3 :
3 |     print("Loop", i)
4 |     i += 1
```

```
Loop 0
```

while - step

In[*]:
```
1 | i = 0
2 | while i < 3 :
3 |     print("Loop", i)
4 |     i += 1
```

```
Loop 0
```

while - step

In[*]:
```
1 | i = 0
2 | while i < 3 :
3 |     print("Loop", i)
4 |     i += 1
```

```
Loop 0
```

while - step

In[*]:
```
1 | i = 0
2 | while i < 3 :
3 |     print("Loop", i)
4 |     i += 1
```

```
Loop 0
Loop 1
```

while - step

In[*]:
```
1 | i = 0
2 | while i < 3 :
3 |     print("Loop", i)
4 |     i += 1
```

```
Loop 0
Loop 1
```

while - step

In[*]:
```
1 | i = 0
2 | while i < 3 :
3 |     print("Loop", i)
4 |     i += 1
```

```
Loop 0
Loop 1
Loop 2
```

while - step

In[*]:
```
1 | i = 0
2 | while i < 3 :
3 |     print("Loop", i)
4 |     i += 1
```

```
Loop 0
Loop 1
Loop 2
```

while

In[ ]:
```
1 | i = 0
2 | while i < 3 :
3 |     print("Loop", i+1)
4 |     i += 1
```

while

In[ ]:
```
1 | i = 0
2 | while i < 3 :
3 |     print("Loop", i+1)
4 |     i += 1
```

```
Loop 1
Loop 2
Loop 3
```

for in range

In[ ]:

```
1 | for i in range(1,4) :
2 |     print(i)
3 |
```

for in range

In[ ]:
```
1 | for i in range(1,4) :
2 |     print(i)
3 |
```

```
1
2
3
```

for...in...

In[ ]:
```
1 | name = "Nick"
2 | for x in name :
3 |     print(x)
```

for...in...

In[ ]:
```
1 | name = "Nick"
2 | for x in name :
3 |     print(x)
```

```
N
i
c
k
```

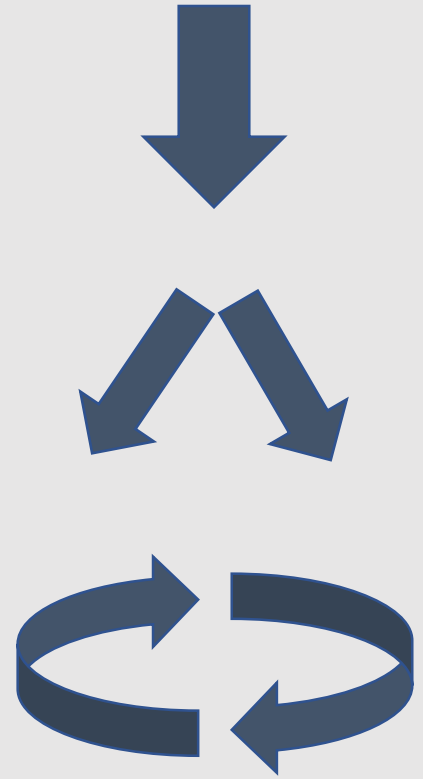
Iteration in C and Java

- Other languages: C, C++, C# and Java feature a do while and the for loop (int i = 0; i < 5; i++) as alternate ways to iterate over between blocks of code.
- The for… in… in Python is equivalent to the for each loop in other languages.
- do while is not present in Python, nor the traditional for loop.
- Increment operator (++) also not in Python because int is a **class**, not a primitive type.

Reminder

Sequence, Selection, Iteration

- **Sequence** mandates that statements be executed in order (line by line)
- **Selection** (conditional) statements will execute a **block of code once** when the condition is true
- **Iteration** allows us to **repeat** statements within a block **whilst** the condition is **true**



Functions in Python

Functions

A function is a block of code that performs a well defined task.

Could be a single line of code, or could be multiple statements which contribute to the achievement of a task.

In our first week, we shall use the main method to achieve basic tasks, such as outputting a message to the screen, or storing the user's name.

Functions

In[ ]:

```
1 | def get_input():
2 |  return input("Please enter your name: ")
3 |
4 |
5 |
```

Functions

In[ ]:

```
1 | def get_input():
2 |  return input("Please enter your name: ")
3 |
4 | name = get_input()
5 | print(name)
```

Functions - run

In[*]:

```
1 | def get_input():
2 |  return input("Please enter your name: ")
3 |
4 | name = get_input()
5 | print(name)
```

Functions - run

In[*]:

```
1 | def get_input():
2 |  return input("Please enter your name: ")
3 |
4 | name = get_input()
5 | print(name)
```

Functions - run

In[*]:

```
1 | def get_input():
2 |  return input("Please enter your name: ")
3 |
4 | name = get_input()
5 | print(name)
```

```
Please enter your name:
```

Functions - run

In[*]:

```
1 | def get_input():
2 |  return input("Please enter your name: ")
3 |
4 | name = get_input()
5 | print(name)
```

```
Please enter your name: Nick
```

Functions - run

In[*]:

```
1 | def get_input():
2 |  return input("Please enter your name: ")
3 |
4 | name = get_input()
5 | print(name)
```

```
Please enter your name: Nick
```

Functions

In[ ]:
```
1 | def get_input():
2 |  return input("Please enter your name: ")
3 |
4 | name = get_input()
5 | print(name)
```

```
Please enter your name: Nick
```

Out[ ]:
```
Nick
```

Functions

In[ ]:

```
1 | def add():
2 |  return x + y
3 |
4 | x = 8
5 | y = 10
6 | add()
```

Functions

In[ ]:
```
1 | def add():
2 |  return x + y
3 |
4 | x = 8
5 | y = 10
6 | add()
```

Out[ ]: 18