

# HKUST Future-Ready Scholars

## Introduction to Game Programming using Python

Part 2

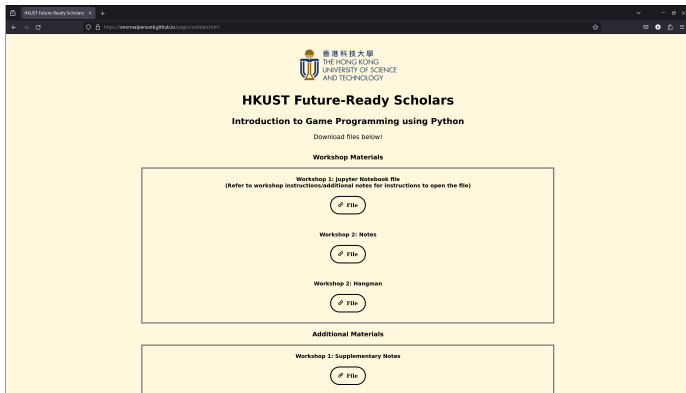
4 May 2024



Login to your Gmail account.

Then head to  
<https://colab.research.google.com/>

All materials today are at:  
<https://bit.ly/ustidpo>



Download all files that belong to **Workshop 2**.

# Jupyter Notebook

Now upload your Jupyter Notebook file with **Files** → **Open Notebook**.



Upload the file **Hangman.ipynb**.

# Using Jupyter Notebook

You can type your code in these blocks. We call these blocks code cells.



```
print("Mum I am in HKUST typing code in a code block")
```

You can run a code cell with the button on the left.



```
print("Hello World!") # Prints "Hello World!"
```

# Summary from Last Workshop

Let's look at what we learnt last time.

# Summary from Last Workshop

## Examples of valid integers

```
a = 5  
b = 1000000  
c = -1984
```

## Examples of valid strings

```
a = "5"  
b = "haha"  
c = 'some words'
```

## Arithmetic Operators

Some basic and commonly-used operators:

+	add	-	minus,
*	multiply	/	divide

# Summary from Last Workshop

## The `print()` statement

```
print(*objects)
```

`*objects` - the things you want to print (put on the screen)

## The `input()` statement

```
input(prompt)
```

where `prompt` is quite literally what it means. It prints the prompt, then returns the value inputted as a string.



# Summary from Last Workshop

## Comparison Operators

There are 6 comparison operators:

Operator	Meaning
==	equal to
>	larger than
>=	larger than or equal to
<	smaller than
<=	smaller than or equal to
!=	not equal to

# Summary from Last Workshop

## if, elif and else

if, elif and else clauses are used to decide whether some code should be executed. Whenever one is fulfilled, all others are ignored.

```
if condition1: # if condition1 is true
    # Do something, ignore all elif and else below

elif condition2: # if condition2 is true
    # Do something, ignore all elif and else below

elif condition3: # if condition3 is true
    # Do something, ignore all elif and else below

else: # if all the conditions above are false
    # Do something
```

# Summary from Last Workshop

## The and logic operator

The and operator makes it so that both conditions have to be fulfilled in order for the code it is under to execute.

## The or logic operator

The or operator makes it so that only 1 of the conditions have to be fulfilled in order for the code it is under to execute.

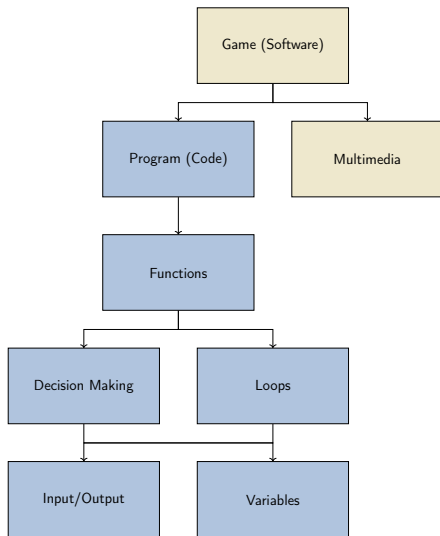
## The not logic operator

The not operator reverses the condition it is attached to.

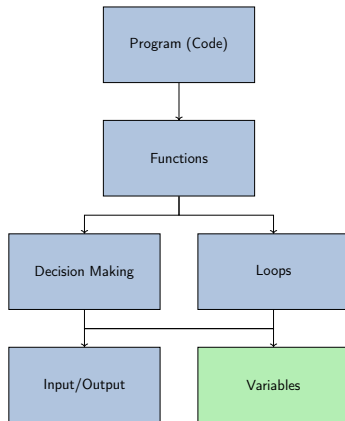
## Multiple logic operators

One can chain multiple logic operators together, but to be safe add brackets () to make sure the condition works as intended.

# World of Game Coding



# Contents



Imagine you have a bunch of variables you want to store. For example, if you have a bunch of people's names.

```
name0 = "Chris Wong"  
name1 = "Desmond Tsoi"  
name2 = "Phoebe Mok"  
name3 = "Nancy Ip"
```

That is annoying to store and access.

What if instead, we store it in the same thing, as a... list?

# Lists

```
names = ["Chris Wong", "Desmond Tsoi",  
         "Phoebe Mok", "Nancy Ip"]  
print(names[0]) # Chris Wong
```

Lists are declared by surrounding the items with [ ], and separating each item with a comma.

We can get the name from a list by getting the corresponding item. How? With list[index].

The first item in the list is the 0<sup>th</sup> item, second is 1<sup>st</sup> item, etc...

We call this zero-indexing.

Note: Some programming languages use one-indexing instead.

If you approach another programming language, be careful.

```
names = ["Chris Wong", "Desmond Tsoi",  
         "Phoebe Mok", "Nancy Ip"]  
print(names[0], names[1], names[2], names[3])  
# Output: Chris Wong Desmond Tsoi Phoebe Mok Nancy Ip
```



Another example:

```
# Indices: 0  1  2  3  4  5
numbers = [0, 1, 1, 2, 3, 5]
print(numbers[0], numbers[1], numbers[2],
      numbers[3], numbers[4], numbers[5])

# Output: 0 1 1 2 3 5

print(numbers)

# Output: [0, 1, 1, 2, 3, 5]
```

One more example in the context of Hangman:

```
# Indices:    0      1      2      3      4      5
word_list = ["p", "y", "t", "h", "o", "n"]
print(word_list[0], word_list[1], word_list[2],
      word_list[3], word_list[4], word_list[5])

# Output: p y t h o n

print(word_list)

# Output: ['p', 'y', 't', 'h', 'o', 'n']
```

To get the length of a list, we can use the `len()` function.

```
numbers = [0, 1, 1, 2, 3, 5]
```

```
print(len(numbers)) # 6
```

```
word_list = ["p", "y", "t", "h", "o", "n"]
```

```
print(len(word_list)) # 6
```

To edit an element of a list, assign the new value to the correct index.

```
numbers = [0, 1, 1, 2, 3, 5]
print(numbers) # [0, 1, 1, 2, 3, 5]
numbers[1] = 100 # Edit the second element (index 1)
print(numbers)
# Output: [0, 100, 1, 2, 3, 5]
```

Another example in the context of Hangman:

```
word_list = ["p", "y", "t", "h", "o", "n"]  
print(word_list) # ['p', 'y', 't', 'h', 'o', 'n']  
word_list[3] = "a" # Edit the fourth element (index 3)  
print(word_list)  
# Output: ['p', 'y', 't', 'a', 'o', 'n']
```

To add an element to the end to a list, we use the `append(value)` list function.

```
numbers = [0, 1, 1, 2, 3, 5]
print(numbers, "length:", len(numbers))
# Output: [0, 1, 1, 2, 3, 5] length: 6
numbers.append(100) # Add 100 to the end of the list
print(numbers, "length:", len(numbers))
# Output: [0, 1, 1, 2, 3, 5, 100] length: 7
```

Another example in the context of Hangman:

```
word_list = ["p", "y", "t", "h", "o", "n"]
print(word_list, "length:", len(word_list))
# Output: ['p', 'y', 't', 'h', 'o', 'n'] length: 6
word_list.append("a") # Edit the fourth element (index 3)
print(word_list, "length:", len(word_list))
# Output: ['p', 'y', 't', 'h', 'o', 'n', 'a'] length: 7
```

We can check if an element is in a list with the `in` operator.

```
numbers = [0, 1, 1, 2, 3, 5]
if 0 in numbers:
    print("0 is in numbers.") # This line is run
else:
    print("0 is not in numbers.")
if 8 in numbers:
    print("8 is in numbers.")
else:
    print("8 is not in numbers.") # This line is run
```



Another example in the context of Hangman:

```
word_list = ["p", "y", "t", "h", "o", "n"]
if "y" in word_list:
    print("y is in the word list.")
else:
    print("y is not in the word list.")
if "a" in word_list:
    print("a is in the word list.")
else:
    print("a is not in the word list.")
```

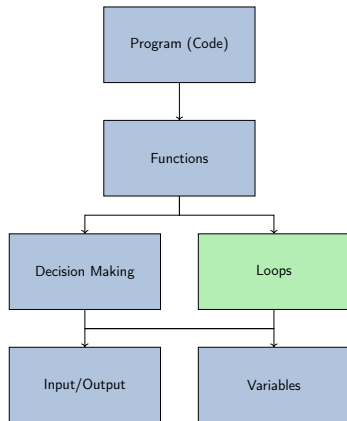
Another example in the context of Hangman:

```
word_list = ["p", "y", "t", "h", "o", "n"]
if "y" in word_list:
    print("y is in the word list.") # This line is run
else:
    print("y is not in the word list.")
if "a" in word_list:
    print("a is in the word list.")
else:
    print("a is not in the word list.")
```

Another example in the context of Hangman:

```
word_list = ["p", "y", "t", "h", "o", "n"]
if "y" in word_list:
    print("y is in the word list.") # This line is run
else:
    print("y is not in the word list.")
if "a" in word_list:
    print("a is in the word list.")
else:
    print("a is not in the word list.") # This line is run
```

# Contents



# Loops

What do you do if you want to do something repeatedly in code?

```
print("Count:", 0)
print("Count:", 1)
print("Count:", 2)
print("Count:", 3)
print("Count:", 4)
print("Count:", 5)
print("Count:", 6)
print("Count:", 7)
print("Count:", 8)
print("Count:", 9)
print("Done.")
```

Let's turn this into a loop.

# Loops - while

Example:

```
i = 0 # Initialising i as 0
while i < 10:
    print("Count:", i)
    i = i + 1
print("Done.")
```

Let's run through it together.

# Loops - while

Example:

```
i = 0
```

```
while i < 10: # i is 0, which is smaller than 10
```

```
    print("Count:", i)
```

```
    i = i + 1
```

```
print("Done.")
```

# Loops - while

Example:

```
i = 0
while i < 10:
    print("Count:", i) # Count: 0
    i = i - 1
print("Done.")
```



# Loops - while

Example:

```
i = 0
```

```
while i < 10:
```

```
    print("Count:", i)
```

```
    i = i + 1 # i goes from 0 to 1, then we go back up
```

```
print("Done.")
```

# Loops - while

Example:

```
i = 0
```

```
while i < 10: # i is 1, which is smaller than 10
```

```
    print("Count:", i)
```

```
    i = i + 1
```

```
print("Done.")
```

# Loops - while

Example:

```
i = 0
while i < 10:
    print("Count:", i) # Count: 1
    i = i + 1
print("Done.")
```

# Loops - while

Example:

```
i = 0
```

```
while i < 10:
```

```
    print("Count:", i)
```

```
    i = i + 1 # i goes from 1 to 2, then we go back up
```

```
print("Done.")
```

# Loops - while

Example:

```
i = 0
```

```
while i < 10: # i is 2, which is smaller than 10
```

```
    print("Count:", i)
```

```
    i = i + 1
```

```
print("Done.")
```

# Loops - while

Example:

```
i = 0
while i < 10:
    print("Count:", i) # Count: 2
    i = i + 1
print("Done.")
```

# Loops - while

Example:

```
i = 0
```

```
while i < 10:
```

```
    print("Count:", i)
```

```
    i = i + 1 # i goes from 2 to 3, then we go back up
```

```
print("Done.")
```

# Loops - while

Example:

```
i = 0
```

```
while i < 10: # i is 3, which is smaller than 10
```

```
    print("Count:", i)
```

```
    i = i + 1
```

```
print("Done.")
```

This goes on and on...



# Loops - while

Example:

```
i = 0
```

```
while i < 10: # i is 9, which is smaller than 10
```

```
    print("Count:", i)
```

```
    i = i + 1
```

```
print("Done.")
```

# Loops - while

Example:

```
i = 0
while i < 10:
    print("Count:", i) # Count: 9
    i = i + 1
print("Done.")
```

# Loops - while

Example:

```
i = 0
```

```
while i < 10:
```

```
    print("Count:", i)
```

```
    i = i + 1 # i goes from 9 to 10, then we go back up
```

```
print("Done.")
```

# Loops - while

Example:

```
i = 0
```

```
while i < 10: # i is 10, which is NOT smaller than 10
```

```
    print("Count:", i)
```

```
    i = i + 1
```

```
print("Done.")
```

# Loops - while

Example:

```
i = 0
while i < 10:
    print("Count:", i)
    i = i + 1
print("Done.") # "Done." is printed
```

# Loops - while

Example:

```
i = 0
while i < 10:
    print("Count:", i)
    i = i + 1
print("Done.")
```

## Indentation

Just like if-clauses, the indentation must be consistent for statements in the loop. This also applies to for loops, which we will get into very soon.

# Loops - while

Example in the context of Hangman:

```
max = 6
wrong_guess = 0
while wrong_guess < max:
    print("You have", max - wrong_guess, "guesses left.")
    # Some more code to decide if the guess is wrong
```

# Loops - for

Example:

```
for i in range(10):  
    print("Count:", i)  
print("Done.")
```

## Python range

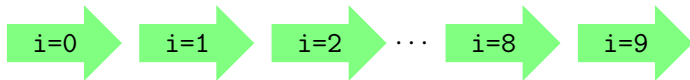
Python range is a thing of mystery. When you do `range(n)`, where `n` is an integer, Python generates a *range* of integers from 0 to `n - 1`.



# Loops - for

Example:

```
for i in range(10):  
    print("Count:", i)  
print("Done.")
```



# Loops - for

```
for i in range(10):  
    print("Count:", i)  
print("Done.")
```

is equivalent to

```
i = 0  
while i < 10:  
    print("Count:", i)  
    i = i + 1  
print("Done.")
```

Both loops go from 0 to 9, and give identical output.

# Loops - for

Another example:

```
for i in range(3):  
    print(i * i) # Print the square, end with a space  
# Output: 0  
#         1  
#         4
```

# Loops - for

Let's combine lists with a for loop.

```
word = ["p", "y", "t", "h", "o", "n"]
```

```
for i in range(len(words)):  
    print(word[i]) # Print word[i]
```

```
# Output: p
```

```
#         y
```

```
#         t
```

```
#         h
```

```
#         o
```

```
#         n
```

This is one way we go through a list.

# Loops - for

Instead of using the index, there is another way to go through a list:

```
word = ["p", "y", "t", "h", "o", "n"]
```

```
for i in word:
```

```
    print(i) # Print the element
```

```
# Output: p
```

```
#         y
```

```
#         t
```

```
#         h
```

```
#         o
```

```
#         n
```

The output is identical to the previous example.

## Lists

Lists are represented with `[ ]` to hold multiple variables, where the  $i^{\text{th}}$  item is at index  $i - 1$ .

## Lists with functions

If a list is called `l`, one can:

- print the list with `print(l)`.
- get the length of `l` with `len(l)`.
- get/edit the element at index `i` with `l[i]`.

## List functions

If a list is called `l`, one can:

- append a value `v` to `l` with `l.append(v)`.
- use the `in` operator to check if a value `v` is in a list.

e.g.: `if v in l:`

## while loops

```
while condition:  
    # Do code
```

Code in the `while` block are run while the condition is fulfilled.  
Do make sure that the `while` loop can be exited.



# Summary

## for loops and range

```
n = 5 # Example
for i in range(n):
    # Do code with each number from 0 to n - 1
```

`range(n)` returns a range of integers that starts from 0 and ends at  $n - 1$ .

## for loops and lists

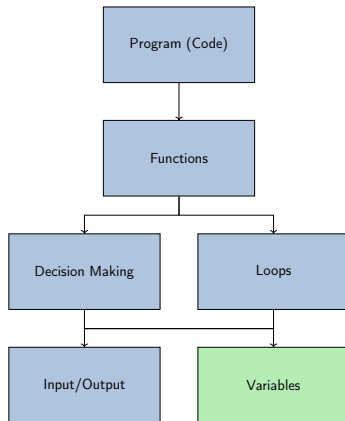
```
l = [...] # A list with items
for i in l:
    # Do code with each item in the list
```

for loops can be directly applied onto lists.

The End  
Thank you!

Here are some additional content that we didn't have time to mention in the workshop.

# Contents



To insert an element to a particular position in a list, we use the `insert()` list function.

The `insert(i, value)` inserts the value at index `i`, and push everything after to the right.

```
numbers = [0, 1, 1, 2, 3, 5]
print(numbers, "length:", len(numbers))
# Output: [0, 1, 1, 2, 3, 5] length: 6
numbers.insert(2, 100) # Add 100 to index 2 of the list
print(numbers, "length:", len(numbers))
# Output: [0, 1, 100, 1, 2, 3, 5] length: 7
numbers.insert(7, 200) # Same as numbers.append(200)
print(numbers, "length:", len(numbers))
# Output: [0, 1, 100, 1, 2, 3, 5, 200] length: 8
```

To remove an element from a list, we use the `remove()` list function. The `remove(value)` function removes the **first** occurrence of value.

```
numbers = [0, 1, 1, 2, 3, 5]
print(numbers, "length:", len(numbers))
# Output: [0, 1, 1, 2, 3, 5] length: 6
numbers.remove(1) # Remove the first occurrence of number 1
print(numbers, "length:", len(numbers))
# Output: [0, 1, 2, 3, 5] length: 5
```

The `reverse()` list function reverses a list's contents.

```
numbers = [0, 1, 1, 2, 3, 5]
print(numbers, "length:", len(numbers))
# Output: [0, 1, 1, 2, 3, 5] length: 6
numbers.reverse() # Reverse the list
print(numbers, "length:", len(numbers))
# Output: [5, 3, 2, 1, 1, 0] length: 6
print(numbers[0])
# Output: 5
```

The `count(item)` list function counts the number of occurrence of `item` in a list.

```
numbers = [0, 1, 1, 2, 3, 5]
```

```
print(numbers.count(1))
```

```
# Output: 2
```

```
print(numbers.count(100))
```

```
# Output: 0
```



The `index(item)` list function finds the index of the first occurrence of `item` in a list.

```
numbers = [0, 1, 1, 2, 3, 5]
```

```
print(numbers.index(1))
```

```
# Output: 1
```

```
print(numbers.index(5))
```

```
# Output: 5
```

```
print(numbers.index(100))
```

```
# Output: No output, error, 100 is not in the list
```

Combining `in` and `list.index()`:

```
numbers = [0, 1, 1, 2, 3, 5]
```

```
if 5 in numbers:
```

```
    print("The index of 5 in the list is", numbers.index(5))
```

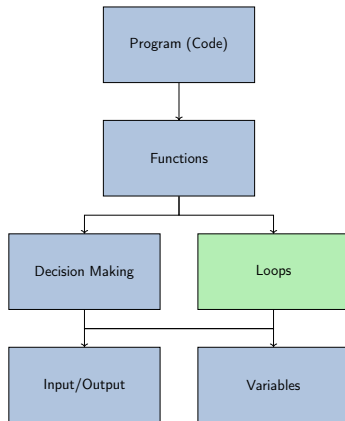
```
# Output: The index of 5 in the list is 5
```

# Lists

The `sort()` list function sorts a list's contents.

```
numbers = [6, 5, 1, 2, 3]
print(numbers, "length:", len(numbers))
# Output: [6, 5, 1, 2, 3] length: 5
print(numbers[0])
# Output: 6
numbers.sort() # Sort the list
print(numbers, "length:", len(numbers))
# Output: [1, 2, 3, 5, 6] length: 5
print(numbers[0])
# Output: 1
```

# Contents



# Loops - while

We can also apply boolean values to while loops.

```
equal_to_5 = False
count = 0
while not equal_to_5:
    if count == 5:
        equal_to_5 = True
    count = count + 1
print("Done.") # "Done." is printed
```

# Summary

The range in Python does not always have to start at 0.

```
for i in range(2, 5):  
    print(i)  
# Output: 2  
#         3  
#         4
```

## Custom range

Given range(a, b), a for loop will iterate from a to b - 1.

## List functions

If a list is called `l`, one can:

- insert a value `v` to `l` at index `i` with `l.insert(i, v)`.
- remove the first occurrence of a value `v` with `l.remove(v)`.
- reverse the list with `l.reverse()`.
- count the occurrence of value `v` with `l.count(v)`.
- get the index of the first occurrence of a value `v` with `l.index(v)`.
- sort the list with `l.sort()`.

# Summary

## Boolean conditions of while

You can apply boolean conditions to while loops.

```
status = True # Or False, or a condition with variables
while status: # Can also add "not"
    # Do something
```

## Custom range

Given range(a, b), a for loop will iterate from a to b - 1.

```
sum = 0
for i in range(100, 102):
    sum = sum + i
print(sum) # Output: 201
```



End of Additional Contents  
Made in L<sup>A</sup>T<sub>E</sub>X  
Last updated: 12 Apr 2024