PYTHON

LISTS

NOR HAMIZAH MISWAN

- Data Science: work with many data points
- Problem
  - height1 = 1.73
  - height2 = 1.68
  - height3 = 1.76
  - height4 = 1.89

Inconvenient!!!

LISTS

- Lists is a collection of items in a particular order
- Can contain any type
- Can contain different type in the same list

```
fam = ["liz", 1.73, "emma", 1.68, "mom", 1.71, "dad", 1.89]
fam
```

```
['liz', 1.73, 'emma', 1.68, 'mom', 1.71, 'dad', 1.89]
```

→ *different data types*

```
fam2 = [["liz", 1.73],
        ["emma", 1.68],
        ["mom", 1.71],
        ["dad", 1.89]]
fam2
```

```
[['liz', 1.73], ['emma', 1.68], ['mom', 1.71], ['dad', 1.89]]
```

→ *a list in a list*

SUBSETTING THE LISTS

- to access the information in the list
- Use "index" to subset the list
- Indexing in python starts from ZERO → **zero indexing**

```
fam = ["liz", 1.73, "emma", 1.68, "mom", 1.71, "dad", 1.89]
fam
```

```
['liz', 1.73, 'emma', 1.68, 'mom', 1.71, 'dad', 1.89]
```

How to access Emma's height?

How about negative indexes?

LIST SLICING

- Creating new list
- Specifying a range using colon ":"

```
fam
```

```
['liz', 1.73, 'emma', 1.68, 'mom', 1.71, 'dad', 1.89]
```

```
fam[3:5]
```

What would the answer be?

How about fam[1:4], fam[:4], fam[5:]?

[start : end]

inclusive exclusive

CHANGE THE VALUE IN THE LISTS

- To change an element/value, use the name of the list followed by the index of the element you want to change, and then provide the new value you want that item to have.

Try:

```
motorcycles = ['honda', 'yamaha', 'suzuki']

print(motorcycles)

motorcycles[0] = 'ducati'

print(motorcycles)
```

ADDING ELEMENTS TO A LIST

- Append elements to the end of list
  - Add new elements to the end of the list
  - Use append() method

```
Try:
motorcycles = ['honda', 'yamaha', 'suzuki']
print(motorcycles)
motorcycles.append('ducati')
print(motorcycles)
```

ADDING ELEMENTS TO A LIST

- Insert elements to the list
  - Add a new element at any position in your list
  - Use insert() method
  - How: specifying the index of the new element and the value of the new item

Try:

```
motorcycles = ['honda', 'yamaha', 'suzuki']
motorcycles.insert(0, 'ducati')
print(motorcycles)
```

REMOVE ELEMENTS FROM A LIST

- If you know the position of the item you want to remove from a list, you can use the del statement.

  Try:

```
motorcycles = ['honda', 'yamaha', 'suzuki']
del motorcycles[0]
print(motorcycles)
del motorcycles[1]
print(motorcycles)
```

REMOVE ELEMENTS FROM A LIST

- If you want to remove the last item in a list, you can use pop() method.
  - However, you can actually use pop() to remove an item in a list at any position by including the index of the item you want to remove in parentheses.

Try:

```
popped_motorcycle = motorcycles.pop()
print(motorcycles)
first_owned = motorcycles.pop(0)
print('The first motorcycle I owned was a ' + first_owned.title() + '.')
```

REMOVE ELEMENTS FROM A LIST

- If you only know the value of the item you want to remove, you can use the remove() method.

Try:

```
motorcycles = ['honda', 'yamaha', 'suzuki', 'ducati']

print(motorcycles)

motorcycles.remove('ducati')

print(motorcycles)
```

ORGANIZE A LIST

- Sorting a list permanently using sort() method
- Sorting a list temporary using sorted() function
- To reverse the original order of a list, you can use the reverse() method.

PYTHON

WORKING WITH LISTS

for LOOP - LOOPING THROUGH AN ENTIRE LIST

- **for** loop
  - do the same action with every item in a list / perform the same statistical operation on every element.
  - the set of steps is repeated once for each item in the list, no matter how many items are in the list.
  - when writing your own for loops that you can choose any name you want for the temporary variable that holds each value in the list. Eg.:

```
for cat in cats:

for item in list_of_items:
```

for LOOP - LOOPING THROUGH AN ENTIRE LIST

- How?

  for variables in set of values:

  do something

- Make sure *do something* is intended – use the tab key for this and variables does not need to be defined before the loop.

for LOOP - LOOPING THROUGH AN ENTIRE LIST

Try:

```
players = ['charles', 'martina', 'michael', 'florence', 'eli']
for player in players:
    print(player)

for player in players:
    print(player.title() + ", that was a great skill!")
    print("I can't wait to see your next match, " + player.title() + ".\n")
print("Thank you, everyone. That was a great game!")
```

for LOOP - LOOPING THROUGH AN ENTIRE LIST

- Using range() function
  - ➢ to generate a series of numbers.
  - ➢ range() function causes Python to start counting at the first value you give it, and it stops when it reaches the second value you provide.
  - ➢ Because it stops at that second value, the output never contains the end value.

Try:

```
for value in range(1,5):
        print(value)
```

for LOOP - LOOPING THROUGH AN ENTIRE LIST

- Using range() function
  - If you want to make a list of numbers, you can convert the results of range() directly into a list using the list() function.
  - range() takes three arguments: range(start, stop, step)

Try:

```
numbers = list(range(1,6))
print(numbers)


squares = []
for value in range(1,11):
        squares.append(value**2)
print(squares)
```

WORKING WITH PART OF A LIST

- Slicing a list

Try:

```
print(players[0:3])

print(players[:4])

print(players[2:])

print(players[:])
```

- Looping through a slice

Try:

```
print("Here are the first three players on my team:")

for player in players[:3]:
        print(player.title())
```

PYTHON

IF STATEMENTS

IF STATEMENTS

- Programming often involves examining a set of conditions and deciding which action to take based on those conditions.
- Python's if statement allows you to examine the current state of a program and respond appropriately to that state.

CONDITIONAL TESTS

- Conditional tests can be evaluated as True or False.
- Checking for equality (==) (case sensitive).

Try:

car = ‘bmw’

car == ‘bmw’

car == ‘audi’

- Checking for inequality (!=)

Try:

car =! ‘audi’

if car != ‘audi’:

print(“That’s not audi!”)

CONDITIONAL TESTS

- Conditional tests can be evaluated as True or False.

- Numerical comparisons.

Try:

Age = 18

Age == 18

Age <= 21

Age > 21

- Checking multiple conditions.

Try:

age_0 = 22

age_1 = 18

age_0 >= 21 or age_1 >= 21

age_0 >= 21 and age_1 >= 17

age_0 >= 21 and age_1 >= 21

CONDITIONAL TESTS

- Conditional tests can be evaluated as True or False.
- Checking value in a list.
  - Use keyword in.

  Try:

  #From list players

  'martina' in players

  'ela' in players

- Checking value not in a list.
  - Use keyword not in.

  Try:

  #From list players

  'martina' not in players

  'ela' not in players

  if 'ela' not in players:

  print(That's not a player")

SIMPLE if STATEMENTS

- The simplest kind of if statements has one test and one action

```
if conditional_test:
    do something
```

```
Try:
age = 19
if age >= 18:
    print("You are old enough to vote!")
```

if-else STATEMENTS

- Python if-else required if we want to take one action when a conditional test passes and a different action in other cases.

```
if conditional_test:
    do something
else:
    do this
```

```
Try:
age = 17
if age >= 18:
    print("You are old enough to vote!")
else:
    Print("Sorry, you are still young!")
```

if-elif-else STATEMENTS

- Python if-elif-else required if we want to take more than two possible situations.
- It runs each conditional test in order until one passes.

```
if conditional_test:
    do something
elif:
    do this
else:
    do this
```

```
Try:
age = 12
if age < 5 :
    print("Your admission cost is RM0.")
elif age < 18 :
    print("Your admission cost is RM5.”)
else:
    print("Your admission cost is RM10.”)
```

TESTING MULTIPLE CONDITIONS

- Python if-elif-else n is powerful, but it's only appropriate to use when you just need one test to pass.
- As soon as Python finds one test that passes, it skips the rest of the tests.
- Hence, you may try multiple if statements.

Try:

```
requested_toppings = ['mushrooms', 'extra
cheese']

if 'mushrooms' in requested_toppings:
    print("Adding mushrooms.")
if 'pepperoni' in requested_toppings:
    print("Adding pepperoni.")
if 'extra cheese' in requested_toppings:
    print("Adding extra cheese.")

print("\nFinished making your pizza!")
```

COMBINE if STATEMENTS WITH LISTS AND for LOOP

- You can do some interesting work when you combine lists and if statements to check for special items in the entire list.

Try:

```
requested_toppings = ['mushrooms', 'green peppers', 'extra cheese']
for requested_topping in requested_toppings:
        if requested_topping == 'green peppers':
                print("Sorry, we are out of green peppers right now.")
        else:
                print("Adding " + requested_topping + ".")
print("\nFinished making your pizza!")
```

PYTHON

DICTIONARIES

DICTIONARY

- Connect pieces of related information.
- Example: store name, age, location and any other info in a dictionary.
- *key-value* pairs
  - use a *key* to access the *value* associated with that key
  - value is a number, a string, a list, or even another dictionary
- A dictionary is wrapped in braces, {}
- Every key is connected to its value by a colon, :
- Individual key-value pairs are separated by commas

DICTIONARY

Try:

```
alien_0 = {'color': 'green', 'points': 5}
print(alien_0['color'])
print(alien_0['points'])
```

- To get the value associated with a key,
  - give the name of the dictionary and then place the key inside a set of square brackets.

ADD NEW *KEY-VALUE* PAIRS

- To add a new key-value pair, give the name of the dictionary followed by the new key in square brackets along with the new value.

Try:

```
alien_0 = {'color': 'green', 'points': 5}
print(alien_0)


alien_0['x_position'] = 0
alien_0['y_position'] = 25
print(alien_0)
```

MODIFY VALUES IN A DICTIONARY

- To modify a value in a dictionary, give the name of the dictionary with the key in square brackets and then the new value you want associated with that key.

Try:

```
alien_0 = {'color': 'green'}
print("The alien is " + alien_0['color'] + ".")

alien_0['color'] = 'yellow‘
print("The alien is now " + alien_0['color'] + ".")
```

REMOVE KEY-VALUE PAIR

- We can use the del statement to completely remove a key-value pair.
- All del needs is the name of the dictionary and the key that you want to remove.

Try:

```
alien_0 = {'color': 'green', 'points': 5}
print(alien_0)

del alien_0['points']
print(alien_0)
```

LOOPING THROUGH A DICTIONARY

- Dictionaries can be used to store information in a variety of ways and therefore, several different ways exist to loop through them.
- You can loop through all of a dictionary's key-value pairs, through its keys, or through its values.
- Several ways of looping style in dictionary:

➢ looping through all key-value pairs

➢ looping through all keys

➢ looping through a dictionary's keys in order

➢ Looping through all values

LOOPING THROUGH A DICTIONARY

- Looping through all key-value pairs
  - to see everything sored in the user's dictionary. How? Use:

    for key, value in dict_name.items(): or for k, v in dict_name.items():

Try:

```
customer = { 'Name': 'John', 'Status': 'Married', 'Hometown': 'Kuala Lumpur'}
for key, value in customer.items():
    print("\nKey: " + key)
    print("Value: " + value)
```

LOOPING THROUGH A DICTIONARY

- Looping through all keys
  - keys() method is useful when you don't need to work with all of the values in a dictionary. How? Use:

for key in dict_name.keys():

```
Try:
print("\nThe following information are required: ")
for info in customer.keys():
    print(info.title())
```

LOOPING THROUGH A DICTIONARY

- Looping through all keys
  - keys() method is useful when you don't need to work with all of the values in a dictionary.
  - Looping through the keys is having the similar behaviour when looping through a dictionary. Hence, these two produce similar results:

```
for key in dict_name.keys():
for key in dict_name:
```

```
Try:
for info in customer:
    print(info.title())
```

LOOPING THROUGH A DICTIONARY

- Looping through a dictionary's keys in order
  - to return items in a certain order is to sort the keys as they're returned in the for loop.
  - Use the sorted() function to get a copy of the keys in order:

```
for key in sorted(dict_name.keys()):
```

```
Try:
print("The following information are required:")
for info in sorted(customer.keys()):
    print(info.title() + " is compulsory!")
```

LOOPING THROUGH A DICTIONARY

- Looping through all values
  - If you are primarily interested in the values that a dictionary contains, you can use the values() method to return a list of values without any keys. How? Use:

```
for value in dict_name.values():
```

```
Try:
print("\nThe details of a customer are listed below: ")
for info in customer.values():
    print(info.title())
```

NESTING

- Nesting is store a set of dictionaries in a list or a list of items as a value in a dictionary.
- You can perform nesting as
  - a set of dictionaries inside a list,
  - a list of items inside a dictionary, or even
  - a dictionary inside another dictionary.

NESTING

- Example of nesting set of dictionaries inside a list:

Try:

```
people1 = {'name': 'John', 'age': '27', 'sex': 'Male'}
people2 = {'name': 'Marie', 'age': '22', 'sex': 'Female'},
people3 = {'name': 'Luna', 'age': '24', 'sex': 'Female', 'married': 'No'}
people4 = {'name': 'Peter', 'age': '29', 'sex': 'Male', 'married': 'Yes'}

peoples = [people1, people2, people3, people4]

for people in peoples:
    print(people)
```

NESTING

- Example of nesting a list of items inside a dictionary:

Try:

```
peoples = {'John': ['27 years old', 'Male'],
          'Marie': ['22 years old', 'Female'],
          'Luna': ['24 years old', 'Female','Not married'],
          'Peter': ['29 years old', 'Male','Married']}

for name, info in peoples.items():
    print("\n" + name.title() + " info's are:")
    for value in info:
        print(value)
```

NESTING

- Example of nesting a dictionary inside another dictionary.

Try:

```
people = {'Albert': {'age': '27',
                     'sex': 'Male'},
          'Enstain':{'age': '22',
                     'sex': 'Female'},
         }
for name, info in people.items():
    print("\n" + name.title() + " info's are:")
    print("Age: " + info['age'] + " years old")
    print("Sex: " + info['sex'])
```

PYTHON

USER INPUT & WHILE LOOPS

PRINTING AND TAKING INPUT

- print(x)
  - x : what do you want to print
  - What it does: shows it on the screen
  - Result: nothing
- input(s)
  - s : what do you want to show on the screen before waiting for input
  - What it does: print s onto the screen and waits for user to type something
  - Result: a string of what the user typed

PRINTING AND TAKING INPUT

- input() function pauses your program and waits for the user to enter some text.
- Once Python receives the user's input, it stores it in a variable to make it convenient for you to work with.

Try:

```
name = input("Please enter your name: ")
print("Hello, " + name + "!")
```

Try:

```
age = input("How old are you? ")
print("You're " + age + " years old!")


age = int(age)
age >= 18
```

while LOOP

- **for** loop takes a collection of items and execute code once for each item in the collection.
- **while** loop
  - run as long as a certain condition is true.
  - only stop running when meet the stopping criteria.
  - How?

  while Boolean expression is True:

  do something

  update variable in Boolean expression

  Note: a while loop stops only if the Boolean eventually become False

while LOOP

- Example of counting a series of numbers

Try:

```
i = 1
while i <= 5:
    print(i)
    i += 1
```

Note: The += operator is shorthand for i = i + 1. Hence, it gives increment i by 1.

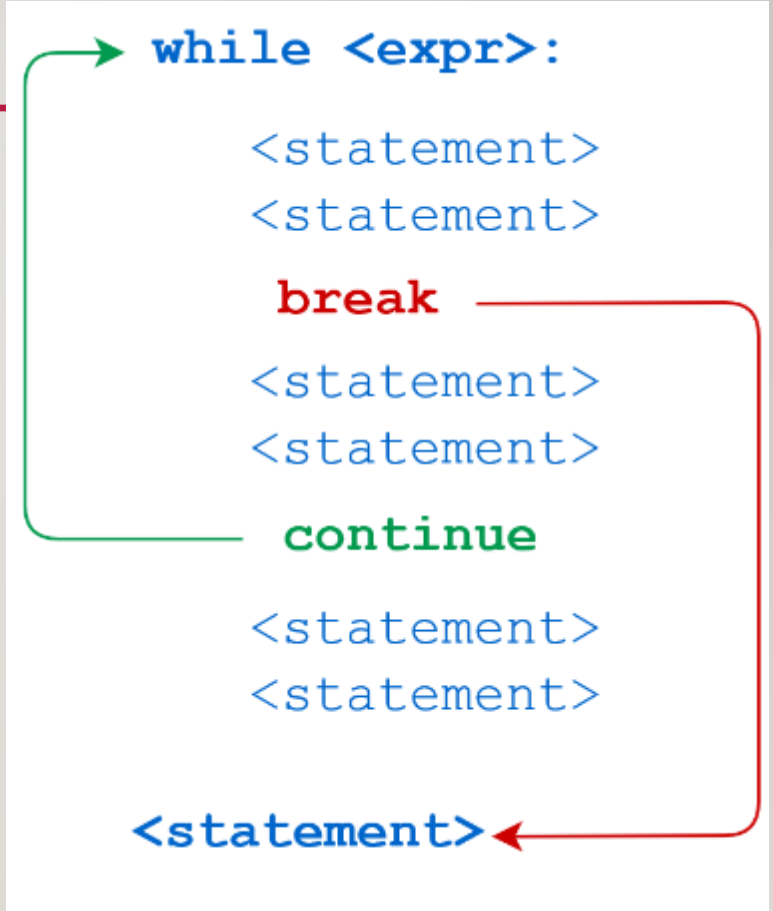
while LOOP

- With the break statement, we can stop the loop even if the while condition is true.

```
Try:
i = 1
while i < 6:
  print(i)
  if (i == 3):
    break
  i += 1
```

- With continue statement, we can stop the current iteration, and continue with the next.

```
Try:
i = 0
while i < 6:
  i += 1
  if i == 3:
    continue
  print(i)
```



while LOOP

- For a program that should run only as long as many conditions are true, you can define one variable that determines whether or not the entire program is active.
- This variable, is called a flag which acts as a signal to the program.
- We can write our programs so they run while the flag is set to True and stop running when any of several events sets the value of the flag to False.

while LOOP

```
Try:
flag = True  # initializing flag with true value
i=10  # initializing i with value 10
# Started a loop with a condition that flag==True
while flag:
    print(i)
    if i==1:
        flag = False # changing the value of flag to False in a way that the loop terminates
before printing 0
    else:
        i-=1
```

while LOOP

- Avoid infinite loops

Try:

```
#This loop is fine!
i = 0
while i < 6:
  print(i)
  i += 1
```

```
#This loop runs forever!
i = 0
while i < 6:
  print(i)
```