SMU | Lyle School of Engineering

CS 1340 Introduction to Computing Concepts

Instructor: Xinyi Ding

Jan 28 2020, Lecture 3

Agenda

- Agenda:
  - Quick review of concepts from last week
  - Dictionaries
  - Operations on dictionaries
  - Boolean expressions, logical operators
  - Control flow, if/else

Variables and Data Types

- Variables
  - containers for storing data values.
  - created the moment you first assign a value to it.

- Common data types
  - Strings: *'this is a string'*
  - Integer: *39*
  - Float: *3.1415926*
  - Boolean: *Ture, False*

Sequence types

- Tuple
  - A simple **immutable** ordered sequence of items
    - Immutable: a tuple cannot be modified once created....
    - Items can be of **mixed types**, including collection types
- List
  - **Mutable** ordered sequence of items of **mixed types**
- String
  - **Immutable**
  - Conceptually very much like a tuple

Lists

- **Mutable** ordered sequence of items of **mixed types**
- Use square brackets []
- Individual elements in the list are separated by commas
- Indexing starts from 0, not 1

```
>>> bicycles = ['trek', 'cannondale', 'redline', 'specialized']
>>> print(bicycles)
['trek', 'cannondale', 'redline', 'specialized']
>>>
```

Dictionaries

- Dictionaries store a mapping between a set of keys and a set of values
  - Keys can be any immutable type
  - Values can be any type
  - Values and keys can be of different types in a single dictionary
- You can define/add/modify/delete key-value pairs of a dictionary

```
>>> credentials = {'alice': 'this is a password', 'carl': 'this is also a password'}
>>> print(credentials['carl'])
this is also a password
>>>
```

Dictionaries

- Creating and accessing dictionaries
  - Use {} to create a dictionary (we use [] for list)
  - Specify the key-value pair if not empty

```
>>> emp_dict = {}
>>> user_info = {'name': 'ethan', 'age': 23, 'address': {'state': 'TX', 'zip': 75206}}
>>> print(user_info['name'])
ethan
>>> print(user_info['address'])
{'state': 'TX', 'zip': 75206}
>>> print(user_info['address']['zip'])
75206
>>>
```

Dictionaries

- Accessing dictionaries

```
>>> user_info = {'name': 'ethan', 'age': 23}
>>> print(user_info['age'])
23
>>> print(user_info[age])
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'age' is not defined
>>>
```

- Key error

```
>>> user_info = {'name': 'ethan', 'age': 23}
>>> print(user_info['name'])
ethan
>>> print(user_info['birthday'])
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
KeyError: 'birthday'
>>>
```

Dictionaries

- Accessing dictionaries use .get() method
  - .get() method allows you to specify a default value if key not exist

```
>>> user_info = {'name': 'ethan', 'age': 23, 'employee_id': 123456}
>>> print(user_info['department'])
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
KeyError: 'department'
>>> print(user_info.get('department', 'IT'))
IT
>>>
```

Dictionaries

- Adding key-value pairs to existing dictionary
  - Dictionaries are unordered
    - New entry might appear anywhere in the output

```
>>> employee_ids = {}
>>> employee_ids['alice'] = 1
>>> print(employee_ids)
{'alice': 1}
>>> employee_ids['carl'] = 2
>>> print(employee_ids)
{'alice': 1, 'carl': 2}
>>>
```

Dictionaries

- Updating dictionaries
  - Keys must be unique
  - Assigning to existing key replace its value

```
>>> employee_ids = {}
>>> employee_ids['alice'] = 1
>>> print(employee_ids)
{'alice': 1}
>>> employee_ids['carl'] = 2
>>> print(employee_ids)
{'alice': 1, 'carl': 2}
>>> employee_ids['alice'] = 3
>>> print(employee_ids)
{'alice': 3, 'carl': 2}
>>>
```

Dictionaries

- Removing dictionaries entries

```
>>> user_info = {'name': 'bob', 'password': '123456789', 'age': 23}
>>> del user_info['age']
>>> print(user_info)
{'name': 'bob', 'password': '123456789'}
>>> user_info.clear()
>>> print(user_info)
{}
>>> a = [1, 2]
>>> del a[1]
>>> print(a)
[1]
>>>
```

Dictionaries

- Useful methods

```
>>> user_info = {'name': 'carl', 'age': 23, 'employee_id': 123456}
>>> user_info.keys()
dict_keys(['name', 'age', 'employee_id'])
>>> type(user_info.keys())
<class 'dict_keys'>
>>> user_info.values()
dict_values(['carl', 23, 123456])
>>> user_info.items()
dict_items([('name', 'carl'), ('age', 23), ('employee_id', 123456)])
>>>
```

- .key() will return a list if you are using python 2.7
- use list(user_info.key()) to get a list, but it is not pythonic
  - type dict_keys is iterable

Demo

Conditional test (Boolean expression)

- Programming often involves examining a set of conditions (conditional test) and deciding which action to take based on those conditions.
- If statement allows you to examine the current state of a program and respond appropriately to that state
- Return Boolean data type: True or False

Conditional test (Boolean expression)

- Check for Equality
  - Most conditional tests compare the current value of a variable to a specific value of interest
  - Note: single = for assignment, == for check equality
  - Case sensitive

```
>>> car = 'audi'
>>> car == 'audi'
True
>>> car == 'bmw'
False
>>> car == 'Audi'
False
>>> car.upper() == 'AUDI'
True
>>> car == 'Audi'.lower()
True
>>>
```

Conditional test (Boolean expression)

- Check for Inequality
  - use !=

```
>>> requested_topping = 'mushrooms'
>>> requested_topping != 'onion'
True
>>>
```

Conditional test (Boolean expression)

- Numerical comparisons

```
>>> age = 18
>>> age == 18
True
>>> answer = 17
>>> answer != 42
True
>>> age = 19
>>> age < 21
True
>>> age <= 21
True
>>> age > 21
False
>>> age >=21
False
>>>
```

Conditional test (Boolean expression)

- Logical operators
  - True if a is True and b is True: a and b
  - True if a is True or b is True: a or b
  - True if a is False: not a

Conditional test (Boolean expression)

- Combine boolean expressions using logical operators

```
>>> age_0 = 22
>>> age_1 = 18
>>> age_0 >= 21 and age_1 >= 21
False
>>> age_1 = 23
>>> age_0 >= 21 and age_1 >= 21
True
>>> (age_0 >= 21) and (age_1 >= 21)
True
>>>
```

```
>>> age_0 = 22
>>> age_1 = 18
>>> age_0 >= 21 or age_1 >= 21
True
>>> age_0 = 18
>>> age_0 >= 21 or age_1 >= 21
False
>>>
```

Conditional test (Boolean expression)

- Other values are treated as equivalent to either True or False when used in conditionals:
  - False: zero, None, empty containers
  - True: non-zero numbers, non-empty objects

Conditional test (Boolean expression)

- Check whether a value is in a list

```
>>> banned_users = ['andrew', 'carolina', 'david']
>>> user = 'marie'
>>> user in banned_users
False
>>> another_user = 'david'
>>> another_user in banned_users
True
>>> user not in banned_users
True
>>> another_user not in banned_users
False
>>>
```

CS 1340 Introduction to Computing Concepts

Instructor: Xinyi Ding

Jan 30 2020, Lecture 4

Agenda

- Agenda:
  - Quick review of concepts from last lecture
  - Control flow (if statement)
  - While/For loops

Control flow

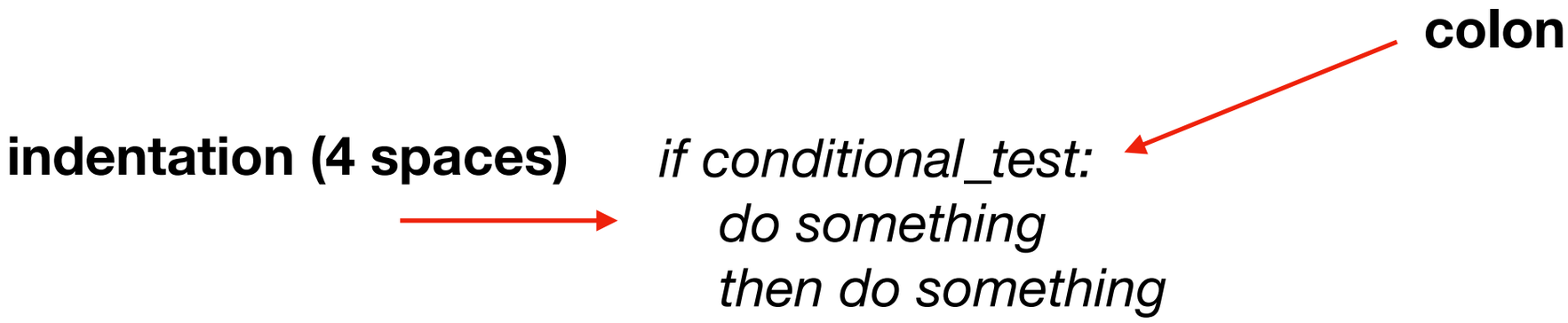
- Python interpreter executes code line by line from top to bottom.
- if statements allow you to take different actions based on different situations

- if statements
  - Use of indentation for blocks
  - Colon(:) after boolean expression.
  - For example:

```
if conditional_test:
    do something
```

if statements

- if statements syntax



- Be careful when using whitespace for indentation
  - use the same number of spaces for indentation. PEP-8 recommends 4 whitespaces. (2 spaces, tab are also valid)
  - you can use any number of spaces in other cases, though valid, but not recommended. (say, x = 5)

if statements

- Simple if statements
  - if the conditional test is True, then execute the following statements, otherwise ignore.

```
1   age = 19
2   if age >= 18:
3       print("You are old enough to vote!")
4       print("Have you registered to vote yet?")
5
6
```

```
1   age = 19
2   if age >= 18:
3   print("You are old enough to vote!")
4   print("Have you registered to vote yet?")
5
6
```

if_statements

```
/Users/xinyi/anaconda/envs/mlearn/bin/python /Users/xinyi/Courses/cs1340/week2/if_statements.py
  File "/Users/xinyi/Courses/cs1340/week2/if_statements.py", line 3
    print("You are old enough to vote!")
        ^
IndentationError: expected an indented block

Process finished with exit code 1
```

if statements

- if-else statements
  - Often, you will want to take one action when a conditional test passes and a different action in all other cases

```
1    age = 17
2    if age >= 18:
3        print("You are old enough to vote!")
4        print("Have you registered to vote yet?")
5    else:
6        print("Sorry, you are too young to vote")
7        print("Please register to vote as soon as you turn 18!")
8
```

else

if_statements ×

```
/Users/xinyi/anaconda/envs/mlearn/bin/python /Users/xinyi/Courses/cs1340/week2/if_statements.py
Sorry, you are too young to vote
Please register to vote as soon as you turn 18!

Process finished with exit code 0
```

if statements

- if-elif-else chain
  - When you need to test more than two possible situations.

```python
age = 12

if age < 4:
    print("Your admission cost is $0")
elif age < 18:
    print("Your admission cost is $5")
else:
    print("Your admission cost is $10")
```

```
if_statements
/Users/xinyi/anaconda/envs/mlearn/bin/python /Users/xinyi/Courses/cs1340/week2/if_statements.py
Your admission cost is $5
```

```python
age = 12

if age < 4:
    price = 0
elif age < 18:
    price = 5
else:
    price = 10

print("Your admission cost is $" + str(price))
```

elif age < 18

```
if_statements
/Users/xinyi/anaconda/envs/mlearn/bin/python /Users/xinyi/Courses/cs1340/week2/if_statements.py
Your admission cost is $5
```

if statements

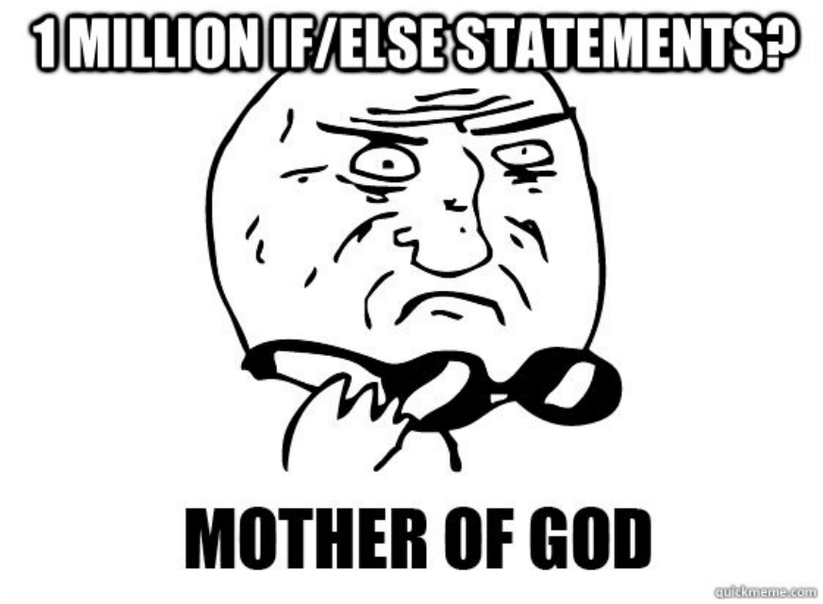
- Using multiple *elif* blocks
  - You can use as many *elif* blocks in your code as you like

```
1   age = 12
2
3   if age < 4:
4       price = 0
5   elif age < 18:
6       price = 10
7   elif age < 65:
8       price = 15
9   else:
10      price = 5
11
12  print("Your admission cost is $" + str(price))
```

else

if_statements

```
/Users/xinyi/anaconda/envs/mlearn/bin/python /Users/xinyi/Courses/cs1340/week2/if_statements.py
Your admission cost is $10
```



if statements

- Omitting the *else* block
  - Python does not require an *else* block at the end of an *if-elif* chain.
  - Sometimes, an else block is useful, sometimes it is clearer to use an additional elif statement that catches the specific condition of interest

```
1   age = 12
2
3   if age < 4:
4       price = 0
5   elif age < 18:
6       price = 10
7   elif age < 65:
8       price = 15
9   elif age >= 65:
10      price = 5
11
12  print("Your admission cost is $" + str(price))
```

elif age >= 65

if_statements

```
/Users/xinyi/anaconda/envs/mlearn/bin/python /Users/xinyi/Courses/cs1340/week2/if_statements.py
Your admission cost is $10
```

if statements

- Testing multiple conditions
  - The *if-elif-else* chain is powerful, but it's only appropriate to use when you just need one test to pass.

```
1   age = 12
2
3   if age < 4:
4       price = 0
5   elif age < 18:
6       price = 10
7   elif age < 65:
8       price = 15
9   else:
10      price = 5
11
12  print("Your admission cost is $" + str(price))
```

else

if_statements

```
/Users/xinyi/anaconda/envs/mlearn/bin/python /Users/xinyi/Courses/cs1340/week2/if_statements.py
Your admission cost is $10
```

Once the conditional test age < 18 is True, it will stop and ignore others

if statements

- Testing multiple conditions
  - Use a series of simple *if* statements with no *elif* or *else* blocks

```
1   requested_toppings = ["mushrooms", "extra cheese"]
2
3   if "mushrooms" in requested_toppings:
4       print("Adding mushrooms.")
5
6   if "pepperoni" in requested_toppings:
7       print("Adding pepperoni.")
8
9   if "extra cheese" in requested_toppings:
10      print("Adding extra cheese.")
11
12  print("Finished making your pizza!")
13
```

if_statements

```
/Users/xinyi/anaconda/envs/mlearn/bin/python /Users/xinyi/Courses/cs1340/week2/if_statements.py
Adding mushrooms.
Adding extra cheese.
Finished making your pizza!
```

if statements

- Testing multiple conditions
  - if we use if-elif-else block

```
1   requested_toppings = ["mushrooms", "extra cheese"]
2
3   if "mushrooms" in requested_toppings:
4       print("Adding mushrooms.")
5
6   elif "pepperoni" in requested_toppings:
7       print("Adding pepperoni.")
8
9   elif "extra cheese" in requested_toppings:
10      print("Adding extra cheese.")
11
12  print("Finished making your pizza!")
13
```

elif "pepperoni" in requested_t...

if_statements ×

```
/Users/xinyi/anaconda/envs/mlearn/bin/python /Users/xinyi/Courses/cs1340/week2/if_statements.py
Adding mushrooms.
Finished making your pizza!
```

if statements

- Other values are treated as equivalent to either True or False when used in conditionals:
  - False: zero, None, empty containers (empty list [])
  - True: non-zero numbers, non-empty objects

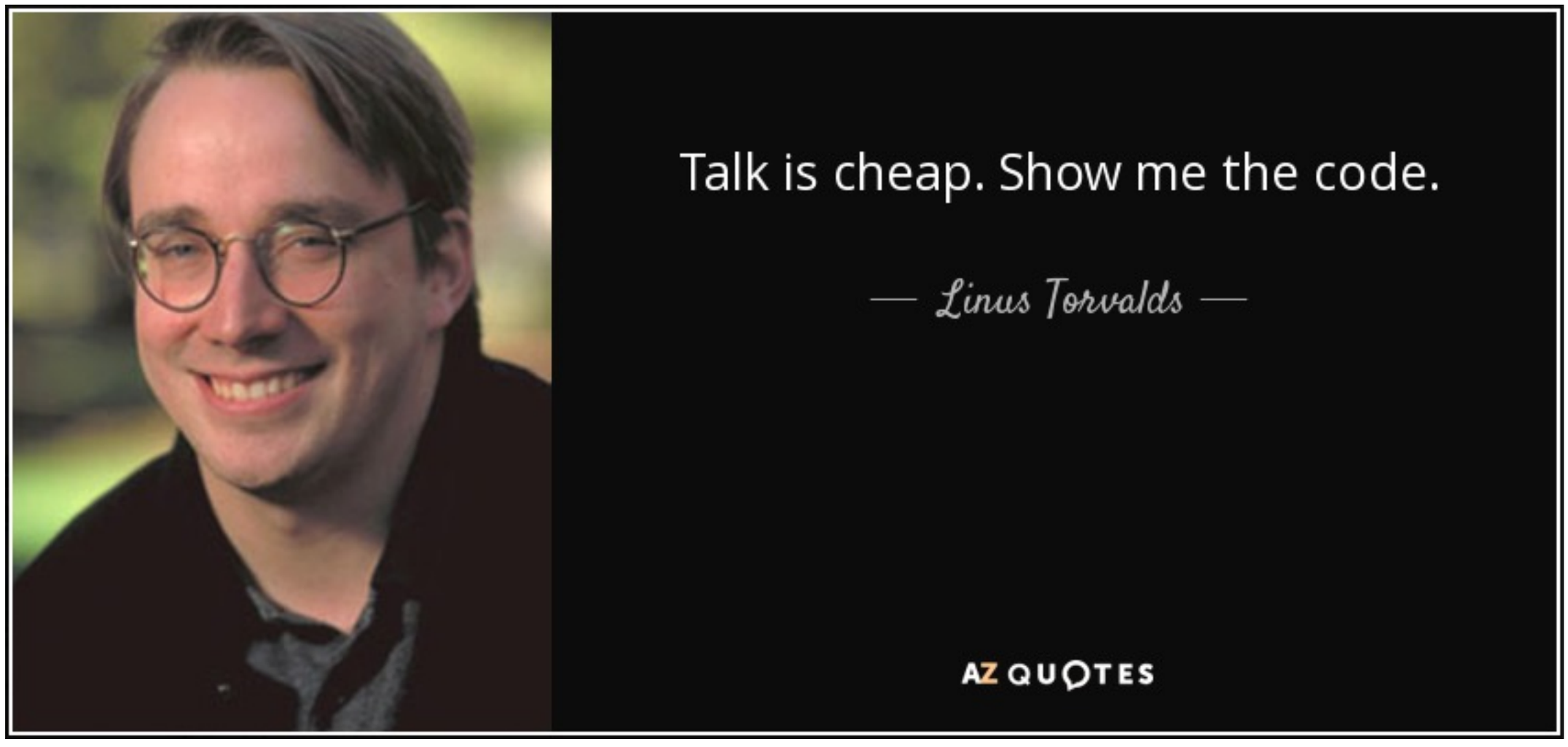
```
1   this_is_a_list = []
2
3   if this_is_a_list:
4       print("this is not an empty list")
5   else:
6       print("this is an empty list")
```

if this_is_a_list

if_statements

```
/Users/xinyi/anaconda/envs/mlearn/bin/python /Users/xinyi/Courses/cs1340/week2/if_statements.py
this is an empty list
```

Demo



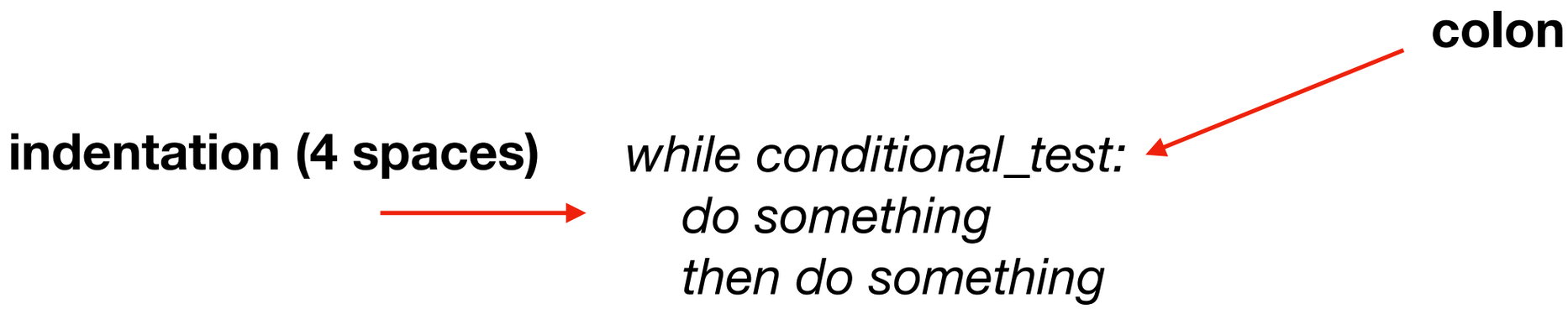
Python loops

- if statements allow you to execute different piece of code based on the different situations (conditional test)
- Loops allow you to execute the same piece of code multiple times

- Python has two primitive loop commands
  - while loops
  - for loops

While loop

- while loop syntax



- It will keep execute the code block as long as the conditional test is true.
  - usually you will need to modify the the values used in the conditional test once some conditions are met

While loop

- A simple example

```
1   current_number = 1
2   while current_number <= 5:
3       print(current_number)
4       current_number += 1
```

```
while_loops
/Users/xinyi/anaconda/envs/mlearn/bin/python /Users/xinyi/Courses/cs1340/week2/while_loops.py
1
2
3
4
5
```

The += operator is shorthand for current_number = current_number + 1

While loop

- Using break to exit a loop
  - To exit a loop immediately without running any remaining code in the loop

```
1       current_number = 1
2       while current_number <= 5:
3           print(current_number)
4           current_number += 1
5           if current_number > 3:
6               break
```

while current_number <= 5 › if current_number > 3

while_loops

```
/Users/xinyi/anaconda/envs/mlearn/bin/python /Users/xinyi/Courses/cs1340/week2/while_loops.py
1
2
3

Process finished with exit code 0
```

While loop

- Using continue in a loop
  - Rather than breaking out of a loop entirely without executing the rest of its code, you can use the continue statement to return to the beginning of the loop based on the result of a conditional test

```
1   current_number = 0
2   while current_number < 10:
3       current_number += 1
4       if current_number % 2 == 0:
5           continue
6       print(current_number)
```

atches and Consoles t_number < 10 › if current_number % 2 == 0

while_loops ×

```
/Users/xinyi/anaconda/envs/mlearn/bin/python /Users/xinyi/Courses/cs1340/week2/while_loops.py
1
3
5
7
9

Process finished with exit code 0
```

While loop

- Avoid infinite loops

```
1    x = 1
2    while x <= 5:
3        print(x)
4        x += 1
```

```
while_loops ×
/Users/xinyi/anaconda/envs/mlearn/bin/python /Users/xinyi/Courses/cs1340/week2/while_loops.py
1
2
3
4
5

Process finished with exit code 0
```

If you omit this, the loop will run forever

```
1    x = 1
2    while x <= 5:
3        print(x)
```

While loop

- Use while loop with lists and dictionaries, example 1

```
1   unconfirmed_users = ["alice", "brain", "candace"]
2   confirmed_users = []
3
4   # Verify each user until there are no more unconfirmed users
5   # Move each verified user into the list of confirmed users.
6
7   while unconfirmed_users:
8       current_user = unconfirmed_users.pop()
9       print("Verifying user:" + current_user)
10      confirmed_users.append(current_user)
11
12  # Display all confirmed users.
13  print(confirmed_users)
```

```
while_loops ×
/Users/xinyi/anaconda/envs/mlearn/bin/python /Users/xinyi/Courses/cs1340/week2/while_loops.py
Verifying user:candace
Verifying user:brain
Verifying user:alice
['candace', 'brain', 'alice']

Process finished with exit code 0
```

While loop

- Use while loop with lists and dictionaries, example 2

```
1  pets = ["dog", "cat", "goldfish", "cat", "dog", "snake", "rabbit"]
2
3  print(pets)
4
5  while "cat" in pets:
6      pets.remove("cat")
7
8  print(pets)
9
```

```
while_loops ×
/Users/xinyi/anaconda/envs/mlearn/bin/python /Users/xinyi/Courses/cs1340/week2/while_loops.py
['dog', 'cat', 'goldfish', 'cat', 'dog', 'snake', 'rabbit']
['dog', 'goldfish', 'dog', 'snake', 'rabbit']

Process finished with exit code 0
```

While loop

- Use while loop with lists and dictionaries, example 3

```
employee_info = {
    "123": {
        "name":"Joe",
        "department": "CS"
    },
    "456": {
        "name": "David",
        "department": "Math"
    },
    "789": {
        "name": "Carl",
        "department": "CS"
    }
}

retired_ids = ["456", "789"]

while retired_ids:
    r_id = retired_ids.pop()
    del employee_info[r_id]

print(employee_info)

```

while retired_ids

while_loops

```
/Users/xinyi/anaconda/envs/mlearn/bin/python /Users/xinyi/Courses/cs1340/week2/while_loops.py
{'123': {'name': 'Joe', 'department': 'CS'}}

Process finished with exit code 0
```

for loop

- For-each is Python's only form of for loop, this is less like the for keyword in other programming languages.
- A for loop steps through each of the items in a collection type (list, dictionary, etc) or any other type of object which is "iterable" (remember when we call .keys() method of a dictionary)
- Often used with lists and dictionaries

colon

indentation (4 spaces)

```
for <each item> in <collection>:
    <statements>
```

for loop

- if <collection> is a list or a tuple, then the loop steps through each element of the sequence

```
1   fruits  = ["apple", "banana", "cherry"]
2   for f in fruits:
3       print(f)
```

```
loops ×
/Users/xinyi/anaconda/envs/mlearn/bin/python /Users/xinyi/Courses/cs1340/week3/loops.py
apple
banana
cherry
```

- if <collection> is a string, then the loop steps through each character of this string

```
1   fruits  = "apple"
2   for f in fruits:
3       print(f)
4
```

```
loops ×
/Users/xinyi/anaconda/envs/mlearn/bin/python /Users/xinyi/Courses/cs1340/week3/loops.py
a
p
p
l
e
```

for loop

- Calculate the sum of a list

```python
a_list = [3, 4, 52, 1, 3, 45, 100, 12]
total_sum = 0
for number in a_list:
    total_sum += number

print(total_sum)
```

```
loops ×
/Users/xinyi/anaconda/envs/mlearn/bin/python /Users/xinyi/Courses/cs1340/week3/loops.py
220
```

- or use built in function

```python
a_list = [3, 4, 52, 1, 3, 45, 100, 12]
total_sum = sum(a_list)

print(total_sum)
```

```
loops ×
/Users/xinyi/anaconda/envs/mlearn/bin/python /Users/xinyi/Courses/cs1340/week3/loops.py
220
```

Making numerical lists

- Many reasons exist to store a set of numbers
  - keep track of the positions of each character in a game
  - keep track of a player's scores
  - store temperatures for data visualization
  - ...

- Lists are ideal for storing sets of numbers, and Python provides a number of tools to help you work efficiently with lists of numbers.

Using the range() Function

- Python's range() function makes it easy to generate a series of numbers. For example

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

for value in range(1, 5)

loops

```
/Users/xinyi/anaconda/envs/mlearn/bin/python /Users/xinyi/Courses/cs1340/week3/loops.py
1
2
3
4
```

Note: range() here gives you 1 through 4, not 5. This behavior is called off-by-one. We have seen this when we used slicing to return a subset of a list

Using range() to Make a List of Numbers

- Call range() does not give you a list, wrap list() around a call to the range() function to get a list.

```
1  numbers = range(1, 5)
2  print(numbers)
3
4  numbers_list = list(range(1, 5))
5  print(numbers_list)
6
7  even_numbers = list(range(2, 11, 2))
8  print(even_numbers)
9
```

loops ×

```
/Users/xinyi/anaconda/envs/mlearn/bin/python /Users/xinyi/Courses/cs1340/week3/loops.py
range(1, 5)
[1, 2, 3, 4]
[2, 4, 6, 8, 10]
```

Using the enumerate() Function

- Use enumerate() function to get the index and elements.

```
1   names = ['alice', 'bob', 'carl']
2   ages  = [18, 32, 22]
3
4   for i, item in enumerate(names):
5       print(i)
6       print(item)
7
```

for i, item in enumerate(names)

loops ×

```
/Users/xinyi/anaconda/envs/mlearn/bin/python /Users/xinyi/Courses/cs1340/week3/loops.py
0
alice
1
bob
2
carl
```

Demo