**2810ICT Software Technologies Lecture Notes**

**Week 1 intro - Python revision**

**Week 1 - introduction**

Ensure study includes Lectures, lecture notes and personal research.

Course inclusions

* Fundamental techniques to develop and implement software systems in industry, including:
  + Scripting techniques
  + Versioning and version control
  + Software testing
* Principles and practice of project management
* Software tools to process , analyse and visualise business data for communication and collaboration purposes.

Staff

* Course convenor: Dr Gervase Tuxworth ([g.tuxworth@griffith.edu.au](mailto:g.tuxworth@griffith.edu.au)) & Dr Thanh Nguyen ([t.nguyen@griffith.edu.au](mailto:t.nguyen@griffith.edu.au))
* Teaching assistant: Owen Xing (O.xing@griffith.edu.au)

Required resources

* Python3
* Pycharm
* PuTTY
* GIT
* Word

Other resources

* Python for Everybody
* Anaconda (python data analysis platform)

**Python notes:**

Variable Types:

Float is a decimal: var = 7.5

Int is an integer: var = 7

Bool is a true or false: var = True

Operators:

|  |  |  |
| --- | --- | --- |
| Number operators | Bool operators | Functional operators |
| + addition | == equal to | And |
| * Subtraction | != not equal to | Not |
| \* multiplication | >= greater than or equal to x | Or |
| \*\* exponent | <= less than or equal to x |  |
| % modulo | < greater, > less |  |

Statements and python functions:

Print(): prints “str” or var or (“str”, var) or (“str” + “str”)

Return(): returns an equation or argument

Comments: # is used to make python comments

Range(a, b, c): gives numbers from a to b with a difference of c between each number

Upper(): capitilises a whole string

Lower(): decapitalises a whole string

Title(): capitalises the first letter of each word in a string

Split(x): splits each word or section at character ‘x’ in a string into a list

Join(): used as “ variable= ' '.join(list)“ turns a list of words into a sentence or string  
Find(): “string.find(‘element’) returns index of start of element  
Strip(): ‘ ‘.strip(list) returns string of list elements stripped of ‘ ‘  
replace(): new\_string = string.replace(‘a’, ‘b’) replaces all ‘a’ with ‘b’ and saves to a new string  
format(): string.format(“blah {paraeter1} blah blah {parameter2}”) allows you to interpolate a string with variables

Split(). String.split(‘thing to split by’) splits string into a list using the entered thing to split by

Defining functions:

Def function(parameters):

Function arguments.

Strings:

Strings are stored as text between quotes “text” or ‘text’

Parts os strings can be called or used by other variables using index notation

e.g. name = “Evey”

age = “25”

user\_name = name[:2]+age

print(user\_name)

will print:

Ev25

Adding quotes in the middle of a string:

String = “I am “Evey” May” #this wont work

String = “I am /”Evey”/ May” #using escape characters will allow this

All string code:

def username\_generator(first\_name, last\_name):

  if len(first\_name) < 3:

    username = first\_name+ last\_name[:4]

  elif len(last\_name) < 4:

    username = first\_name[:3] + last\_name

  else:

    username = first\_name[:3] + last\_name[:4]

  return username

def password\_generator(username):

  password = ""

  for index in range(0, len(username)):

    password += username[index-1]

  return password

Lists:

My\_list = [items separated by commas]

My\_list.append(item) (adds item to end of the list)

My\_list2 = My\_list + [1, 3] (adds 1 and 3 to the end of My\_list)

My\_list3 = zip(list1, list2) (makes a list of lists [[list1a, list2a], [list1b, list2b]]

Var = len(my\_list) (gives amount of items in the list)

A chunk of list elements can be assigned to a new variable using index:

Elementx = list[x]

Start = list[0:x] or list[:x]

Middle = list[x:y]

End = list[x:y] or list[x:]

List.sort() sorts list in alphabetical order

Sorted(list) takes the alphabetically sorted list and assigns it a value

List.Count(item) counts the amount of occurrences of item in List

Tuples:

Very similar to lists however are immutable and cannot be added to, changed or edited once defined.

My\_tuple = (items)

If creating a one element tuple, a comma must go after the assignment (e.g. one\_element\_tuple = (‘element’,)

Assigning variables in (or unpacking) a tuple:

My\_tuple = (“evey”, 25, “tutor”)

Name, age, job = my\_tuple

This will assign name = “evey”, age = 25 and job = “tutor”

Loops:

In lists a *for loop* can be used to print each list item without the constraints of its list:

e.g. girls\_names = [Emma, May, Bella, June]

for names in girls\_names:

print(names)

console will print:

Emma

May

Bella

June

For loops may also be used in conjunction with range() to print multiple lines of the same thing.

promise = "I will not chew gum in class"

for i in range(5):

   print(promise)

console will print:

I will not chew gum in class

I will not chew gum in class

I will not chew gum in class

I will not chew gum in class

I will not chew gum in class

Adding lists together:

students\_period\_A = ["Alex", "Briana", "Cheri", "Daniele"]

students\_period\_B = ["Dora", "Minerva", "Alexa", "Obie"]

for i in students\_period\_A:

   students\_period\_B.append(i)

If statements in for loops:

e.g. 1

dog\_breeds\_available\_for\_adoption = ['french\_bulldog', 'dalmatian', 'shihtzu', 'poodle', 'collie']

dog\_breed\_I\_want = 'dalmatian'

for breed in dog\_breeds\_available\_for\_adoption:

   print(breed)

  if breed == dog\_breed\_I\_want:

     break #break stops the loop once the breed is found

print('They have the dog I want!')

e.g.2

ages = [12, 38, 34, 26, 21, 19, 67, 41, 17]

for age in ages:

    if age >= 21:

     print(age)

For loops inside for loops:

sales\_data = [[12, 17, 22], [2, 10, 3], [5, 12, 13]]

scoops\_sold = 0

for locations in sales\_data:

   for scoops in locations:

     scoops\_sold += scoops #adds each number to the scoops\_sold variable

print(scoops\_sold)

While loops:

all\_students = ["Alex", "Briana", "Cheri", "Daniele", "Dora", "Minerva", "Alexa", "Obie", "Arius", "Loki"]

students\_in\_poetry = []

students = 0

while len(students\_in\_poetry) < 6:

students = all\_students.pop() #.pop() takes item from the end of the list

   students\_in\_poetry.append(students)

print(students\_in\_poetry)

List comprehension:

While we can write an entire loop and append to a new list such as:

words = ["@coolguy35", "#nofilter", "@kewldawg54", "reply", "timestamp", "@matchamom", "follow", "#updog"]  
usernames = []  
  
for word in words:

if word[0] == '@':  
    usernames.append(word)

We can also use list comprehension, which is a single line piece of code that

does the same thing.

Usernames = [word for word in words if word[0] == ‘@’]

Another example:

celsius = [0, 10, 15, 32, -5, 27, 3]

fahrenheit = [temp \* 9/5 + 32 for temp in celsius]

print(fahrenheit)

Putting it all together:

# regular for loop method

single\_digits = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

squares = []

for digits in single\_digits:

print(digits)

squares.append(digits\*\*2)

print(squares)

# list comprehension method

cubes = [digits\*\*3 for digits in single\_digits]

print(cubes)

**Python review**

Variable Types:

* Float is a decimal: var = 7.5
* Int is an integer: var = 7
* Bool is a true or false: var = True
* Str is a string denoted by “”

Variables are named values stored in memory.

Statements assign a value to a name to create a variable.

Operators:

|  |  |
| --- | --- |
| **Operator Name** | **Operator** |
| Addition | + |
| Subtraction | - |
| Multiplication | \* |
| Exponent | \* |
| Modulo | % |
| Division | / |
| Int division | // |
| Equal to | == |
| Not equal to | != |
| Greater or equal | X <= |
| Less or equal | X >= |
| Greater than | X< |
| Less than | X> |
| Functional and | And |
| Functional not | Not |
| Functional or | Or |

Loops

Standard loops

|  |  |  |
| --- | --- | --- |
| **Loop** | **Example** | **Explanation** |
| While loop | While condition:  Loop body statement(s) | Whilst the condition is true, code will execute the body statement. |
| For loop | For variable-name in sequence:  Loop body statement(s) | Variable will be assigned item in the sequence for each execution of the body statement. |

For loops with ranges

|  |  |  |
| --- | --- | --- |
| **Constructor call** | **Range** | **Explanation** |
| Range(5) | 0, 1, 2, 3, 4 | Range starts with 0 ends with 5 (non-inclusive) |
| Range(1, 5) | 1, 2, 3, 4 | Range starts at 1 (inclusive) ends at 5 (non-inclusive) |
| Range(0, 10, 3) | 0, 3, 6, 9 | Range starts at 0 (inclusive) and ends at 10 (non-inclusive) counting in blocks of 3 |
| Range(10, 0, 2) | 10, 8, 6, 4, 2 | Range starts at 10 (inclusive) and ends at 0 (non-inclusive) counting down in blocks of 2 |

If and elif statements

|  |  |
| --- | --- |
| **Statement** | **Explanation** |
| If condition:  Action(s)  Elif condition:  Action(s)  Elif condition:  Action(s) Else:  Action(s) | Selection through alternate actions based on a conditional clause. |

Break & continue statements

Break statement is used to terminate a loop (break)

Continue statements terminate the current execution of the loop body, but not the loop (skips to the next iteration).

Exception handling

* Not to be overused (do not use when exception is obvious)
* Used for handling errors like trying to open a file that may not exist.