INFO1110 Final Exam Revision Notes

Statement:
All content covered in this revision notes is a summary of the materials given in INFO1110 taught in the University of Sydney and plus some extra notes (e.g. definition of terminologies) from the internet). Distributed under fair use and copyright etc. I DO NOT condone and am not liable for any academic malpractice that may eventuate the use of this paper.

Outline

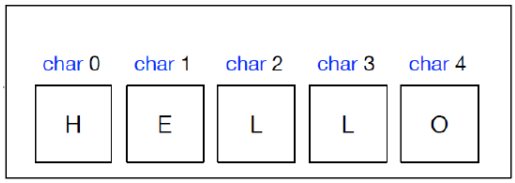
Python Basics (e.g. variables, statements)

Types of Variables can be:
String, Boolean, Integer, Float, List, Tuples, Dictionaries

String

To initialize a string variable, we use single quotes ' double quotes "

```
UnitCode = 'INFO1110'
Language = "Python"
message = """This unit trains students with software development process,
including skills of testing and debugging. """
message = "This unit trains students with software development process, \
including skills of testing and debugging. "
```



and triple quotes """ or ''' to denote literal strings.

Statements typically ends with a new line. If we use multi-line statements, we use \ like the above code.

Importantly, there are several operations you must know about string:
(string slices (index), split, replace, operations, format)

```
s = 'Learn Python'
print(s[1])
print(s[:6])
print(s[3:])
new_s = s.split(' ')
print(new_s)
```

```
new_s = s.replace('n','m')
print(new_s)
new_s *= 2
print(new_s)
new_s += '!!!'
print(new_s)
```

```
sem = 1
time = 'half'
print('We have learned coding for {} semester, i.e. {} of a year.'.format(sem,time))
```

More Formatting of string you may need to know:

```
nums = np.zeros(3, dtype=np.uint8)
print(nums)

n1 = np.zeros(6) # 1D array of length 3 all values 0
print(n1)
```

```
[0 0 0]
[0. 0. 0. 0. 0. 0.]
```

np.zeros(number) or np.zeros(number, dtype=...) gives you an array of zero. Dtype is optional, its default is np.float64.

```
nums = np.array([4396, -666, 23333, 1110], dtype = np.int32)

print(type(nums))
print(nums)
```

```
<class 'numpy.ndarray'>
[ 4396  -666 23333  1110]
```

np.array(object, dtype=...) is used to create an array. Dtype is optional as well.

See more examples below.

```
>>> np.array([1, 2, 3])
array([1, 2, 3])
```

```
>>> np.array([1, 2, 3.0])
array([ 1.,  2.,  3.])
```

This happens when dtype is not set.

When using uint8 dtype which has maximum of 255:

```
nums = np.array ([252,253,254,255,256,257] , dtype = np.uint8)
i = 0
while i < len(nums):
    nums[i] += 1
    i += 1
    print(nums)
```

```
[253 253 254 255   0   1]
[253 254 254 255   0   1]
[253 254 255 255   0   1]
[253 254 255   0   0   1]
[253 254 255   0   1   1]
[253 254 255   0   1   2]
```

```
nums = np.random.random(5).astype ('float64')
print(nums)
```

```
[0.51910645 0.56410163 0.19313186 0.60384721 0.39282849]

[0.06635407 0.87252999 0.17941049 0.90377169 0.61101615]

[0.4834744  0.91812263 0.62066298 0.0287349  0.17499245]
```

Np.random.random() gives you a float number in the range of [0.0, 1.0).

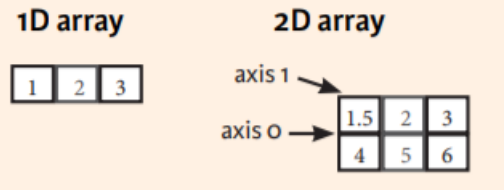
astype() is used to set dtype.

```
nums = np.linspace(2.0, 3.0, num=5)
print(nums)
```

np.linspace(start, stop, num) returns evenly spaced numbers over a specified interval. Num is optional, and its default is 50.

```
>>> np.array([[1, 2], [3, 4]])
array([[1, 2],
       [3, 4]])
```

This a 2D array.



Which of the following is a correct traversal? Or both are correct? Or neither is correct?

Class iterator and generator

Iteration is a general term for taking each item of something, one after another.
Class iterator is similar to list. Here's a quick look how we iterate elements in a list:

Difference: **iterable** and **iterator**

An **iterable** is an object that has an __iter__ method which returns an **iterator.** Thus, an iterable is an object that you can get an **iterator** from.
An **iterator** is an object with __next__ method.

Whenever you use a for loop, or map, or a list comprehension, etc. in Python, the __next__ method is called automatically to get each item from the iterator, thus going through the process of iteration.

```
# define a list
my_list = [0, 2, -1, 9]

# get an iterator using iter()
my_iter = iter(my_list)

## iterate through it using next()
print(next(my_iter))

print(next(my_iter))
## next(obj) is same as obj.__next__()

print(my_iter.__next__())
print(my_iter.__next__())

## This will raise error, no items left
next(my_iter)
```

```
0
2
-1
9

---------------------------------------------------------------------------
StopIteration                             Traceback (most recent call last)
<ipython-input-7-8fad4afbbcb9> in <module>()
     15
     16 ## This will raise error, no items left
---> 17 next(my_iter)

StopIteration:
```

Now, let's look at a Class Iterator Example.

```
class BoundedRepeater:
    def __init__(self, value, max_repeats):
        self.value = value
        self.max_repeats = max_repeats
        self.count = 0

    def __iter__(self):
        return self

    def __next__(self):
        if self.count >= self.max_repeats:
            raise StopIteration
        self.count += 1
        return self.value
```

```
repeater = BoundedRepeater('Hello', 3)
for item in repeater:
    print(item)
```

```
Hello
Hello
Hello
```

```
repeater = BoundedRepeater('Hello', 3)
iterator = iter(repeater)
while True:
    try:
        item = next(iterator)
    except StopIteration:
        break
    print(item)
```

```
Hello
Hello
Hello
```

For loop can do automatic __next__() as the code above. Use either __next__() or next() we can perform iteration.

Note: be careful with how you write __iter__() and __next__().