COMP10001

Week 5

Methods, iteration and loops

What is a method? How are methods different/similar to functions?

What is a method? How are methods different/similar to functions?

- Like functions, methods run some pre-defined code to achieve a task
- Both are called with brackets, containing arguments
- Methods are “attached” to an object with a full stop:
  - `function_name(arguments, …)` vs `object.method_name(arguments, …)`
- So methods can do fun things like edit the object they’re called on in-place!

Let's do an exercise!

Let `s="Computing is FUN!"`. What happens when we apply the methods below?

```
s.isupper()
```

```
s.upper()
```

```
s.endswith("fun!")
```

```
s.count("n")
```

```
s.strip("!")
```

```
s.replace("i", "!")
```

Let's do an exercise!

Let `s="Computing is FUN!"`. What happens when we apply the methods below?

`s.isupper()`

False

`s.upper()`

"COMPUTING IS FUN!"

`s.endswith("fun!")`

True

`s.count("n")`

2

`s.strip("!")`

"Computing is FUN"

`s.replace("i", "!")`

"Comput!ng !s FUN!"

What is the difference between a list and a tuple?

What is the difference between a list and a tuple?

- Lists are mutable
- Tuples are not!

What is the difference between a list and a tuple?

- Lists are mutable
- Tuples are not!
- We can make changes to mutable items ***“in-place”***
  - E.g. lst += [3]

What is the difference between a list and a tuple?

- Lists are mutable
- Tuples are not!
- We can make changes to mutable items ***“in-place”***
  - E.g. lst += [3]
- Lists initialised with square brackets
  - E.g. lst = [1,2,3]
- Tuples initialised with rounds brackets
  - E.g. tup = (1,)
  - Or the tuple() function!

How do we add and remove items from a list?

How do we add and remove items from a list?

- .append(item),
- .insert(index, item),
- .extend(lst)

- .pop(index)
- del
- .remove(item)
- .clear()

Let's do an exercise!

Let `lst=[2, ("green", "eggs", "ham"), False]`. What happens when we use the expressions below?

```
lst[2]
```

```
lst.append(5); print(lst)
```

```
lst[1][-2]
```

```
lst.pop(2); print(lst)
```

```
lst[1][-2][:3]
```

```
lst.reverse(); print(lst)
```

Let's do an exercise!

Let `lst=[2, ("green", "eggs", "ham"), False]`. What happens when we use the expressions below?

```
lst[2]
```

False

```
lst[1][-2]
```

"eggs"

```
lst[1][-2][:3]
```

True

```
lst.append(5); print(lst)
```

[2, ("green", "eggs", "ham"), False, 5]

```
lst.pop(2); print(lst)
```

False
[2, ("green", "eggs", "ham")]

```
lst.reverse(); print(lst)
```

[False, ("green", "eggs", "ham"), 2]

What is “iteration” in programming? Why do we need it?

What is “iteration” in programming? Why do we need it?

- Iteration = executing a section of code repeatedly
  - often with a small change each time

What is “iteration” in programming? Why do we need it?

- Iteration = executing a section of code repeatedly
  - often with a small change each time
- We need to do this all the time in programming!
- Writing the same instructions over and over again is inefficient! Unreadable! Error-prone!

What are the two types of loop in Python? How do we structure them?

What are the two types of loop in Python? How do we structure them?

- For loops

```
For <loop variable> in <sequence>:
    Do something with the loop variable
    ...
```

What are the two types of loop in Python? How do we structure them?

- For loops

```
For <loop variable> in <sequence>:
    Do something with the loop variable
    …
```

- While loops

```
While <condition>:
    Do something
    …
```

How are the two types of loops different?
When should we use one over the other?

How are the two types of loops different?
When should we use one over the other?

- For loop iterates over a sequence:
  - finite/controlled number of iterations

How are the two types of loops different?
When should we use one over the other?

- For loop iterates over a sequence:
  - finite/controlled number of iterations
- While loop iterates until truth of condition changes:
  - infinite/uncontrolled number of iterations

How are the two types of loops different?
When should we use one over the other?

- For loop iterates over a sequence:
  - finite/controlled number of iterations
- While loop iterates until truth of condition changes:
  - infinite/uncontrolled number of iterations

- For loops are great when we know in advance how many times we need our loop to run
- While loops are great when we don’t know!

Can we always convert a for into a while and vice versa?

Can we always convert a for into a while and vice versa?

- For can always be converted into a while
- Converting a while into a for is not always possible!

Can we always convert a for into a while and vice versa?

- For can always be converted into a while
- Converting a while into a for is not always possible!
- To convert a loop, identify:

Can we always convert a for into a while and vice versa?

- For can always be converted into a while
- Converting a while into a for is not always possible!
- To convert a loop, identify:
  a. What the loop variable is initialised to

Can we always convert a for into a while and vice versa?

- For can always be converted into a while
- Converting a while into a for is not always possible!
- To convert a loop, identify:
  a. What the loop variable is initialised to
  b. How the loop variable is incremented/changed during each iteration

Can we always convert a for into a while and vice versa?

- For can always be converted into a while
- Converting a while into a for is not always possible!
- To convert a loop, identify:
  a. What the loop variable is initialised to
  b. How the loop variable is incremented/changed during each iteration
  c. When the loop should terminate

Can we always convert a for into a while and vice versa?

- For can always be converted into a while
- Converting a while into a for is not always possible!
- To convert a loop, identify:
  a. What the loop variable is initialised to
  b. How the loop variable is incremented/changed during each iteration
  c. When the loop should terminate

```
lst = [“a”,”b”,”c”]
for letter in lst:
    print(letter + “!”)
```

Can we always convert a for into a while and vice versa?

- For can always be converted into a while
- Converting a while into a for is not always possible!
- To convert a loop, identify:
  a. What the loop variable is initialised to
  b. How the loop variable is incremented/changed during each iteration
  c. When the loop should terminate

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

Can we always convert a for into a while and vice versa?

- For can always be converted into a while
- Converting a while into a for is not always possible!
- To convert a loop, identify:
  a. What the loop variable is initialised to
  b. How the loop variable is incremented/changed during each iteration
  c. When the loop should terminate

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

i	j	i + j

Can we always convert a for into a while and vice versa?

- For can always be converted into a while
- Converting a while into a for is not always possible!
- To convert a loop, identify:
  a. What the loop variable is initialised to
  b. How the loop variable is incremented/changed during each iteration
  c. When the loop should terminate

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

i	j	i + j
1	1	2

Can we always convert a for into a while and vice versa?

- For can always be converted into a while
- Converting a while into a for is not always possible!
- To convert a loop, identify:
  a. What the loop variable is initialised to
  b. How the loop variable is incremented/changed during each iteration
  c. When the loop should terminate

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

i	j	i + j
1	1	2
2	1	3

Can we always convert a for into a while and vice versa?

- For can always be converted into a while
- Converting a while into a for is not always possible!
- To convert a loop, identify:
  a. What the loop variable is initialised to
  b. How the loop variable is incremented/changed during each iteration
  c. When the loop should terminate

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

i	j	i + j
1	1	2
2	1	3
3	2	5

Exercises!

What is the output of the following snippets?

```
i = 2
while i < 8:
    print(f"The square of {i} is {i * i}")
    i = i + 2
```

```
MIN_WORD_LEN = 5
long_words = 0
text = "There once lived a princess"
for word in text.split():
    if len(word) >= MIN_WORD_LEN:
        print(word, "is too long!")
        long_words += 1
print(long_words, "words were too long")
```

```
i = 0
colours = ("pink", "red", "blue", "gold", "red")
while i < len(colours):
    if colours[i] == "red":
        print("Found red at index", i)
    i += 1
```

```
for ingredient in ("corn", "pear", "chilli", "fish"):
    if ingredient.startswith('c'):
        print(ingredient, "is delicious!")
    else:
        print(ingredient, "is not!")
```

Rewrite the snippets with an alternative loop command!

```
i = 2
while i < 8:
    print(f"The square of {i} is {i * i}")
    i = i + 2
```

```
MIN_WORD_LEN = 5
long_words = 0
text = "There once lived a princess"
for word in text.split():
    if len(word) >= MIN_WORD_LEN:
        print(word, "is too long!")
        long_words += 1
print(long_words, "words were too long")
```

```
i = 0
colours = ("pink", "red", "blue", "gold", "red")
while i < len(colours):
    if colours[i] == "red":
        print("Found red at index", i)
    i += 1
```

```
for ingredient in ("corn", "pear", "chilli", "fish"):
    if ingredient.startswith('c'):
        print(ingredient, "is delicious!")
    else:
        print(ingredient, "is not!")
```

What's wrong with this code? How should we fix it?

```
def largest_num(nums):
    maxnum = nums[0]
    for num in nums:
        if num > maxnum:
            maxnum = num
            return maxnum

print(largest_num([1, 2, 3]))
```

Let’s jump in breakout rooms for Ex. 3

Ex. 3 solutions

```
for i in range(5):
    print(i**2)
```

Ex. 3 solutions

```
for i in range(5):
    print(i**2)
```

```
0
1
4
9
16
```

Ex. 3 solutions

```
for ingredient in ("carrot", "lettuce", "cucumber", "tomato"):
    if ingredient.startswith('c'):
        print(ingredient, "is delicious")
    else:
        print(ingredient, "is tasty")
```

Ex. 3 solutions

```
for ingredient in ("carrot", "lettuce", "cucumber", "tomato"):
    if ingredient.startswith('c'):
        print(ingredient, "is delicious")
    else:
        print(ingredient, "is tasty")
```

```
carrot is delicious
lettuce is tasty
cucumber is delicious
tomato is tasty
```

Ex. 3 solutions

```
i = 0
colours = ("olive", "red", "violet", "turquoise", "red", "red", "amber")
while i < len(colours):
    if colours[i] == "red":
        print("Found red at index", i)
    i += 1
```

Ex. 3 solutions

```
i = 0
colours = ("olive", "red", "violet", "turquoise", "red", "red", "amber")
while i < len(colours):
    if colours[i] == "red":
        print("Found red at index", i)
    i += 1
```

```
Found red at index 1
Found red at index 4
Found red at index 5
```

Ex. 3 solutions

```
MIN_WORD_LEN = 4
long_words = 0
text = "Once upon a time in a land far away lived a princess"
for word in text.split():
    if len(word) > MIN_WORD_LEN:
        print(word, "is too long!")
        long_words += 1
print(long_words, "words were too long")
```

Ex. 3 solutions

```
MIN_WORD_LEN = 4
long_words = 0
text = "Once upon a time in a land far away lived a princess"
for word in text.split():
    if len(word) > MIN_WORD_LEN:
        print(word, "is too long!")
        long_words += 1
print(long_words, "words were too long")
```

```
lived is too long!
princess is too long!
2 words were too long
```

Last exercise for today!
Do the following snippets do the same thing?
What are their advantages/disadvantages?

```
print("We need some saws")
print("We need some hammers")
print("We need some cogs")
print("We need some nails")
```

```
def get_str(part):
    return f"We need some {part}"

print(get_str("saws"))
print(get_str("hammers"))
print(get_str("cogs"))
print(get_str("nails"))
```

```
def get_str(part):
    return f"We need some {part}"

parts = ("saws", "hammers", "cogs", "nails")

for part in parts:
    print(get_str(part))
```

Coding problems

Write a function which takes an integer input n and prints the thirteen times tables from 1 * 13 until n * 13.

Write a function which takes a tuple of strings and returns a list containing only the strings which contain at least one exclamation mark or asterisk symbol.

Coding problems

Write a function which takes an integer input n and prints the thirteen times tables from 1 * 13 until n * 13.

Write a function which takes a tuple of strings and returns a list containing only the strings which contain at least one exclamation mark or asterisk symbol.

```
def thirteen_table(num):
    for i in range(1, num+1):
        print(f"{i} * 13 = {i * 13}")
```

```
def fun_filter(words):
    out = []
    for word in words:
        if '!' in word or '*' in word:
            out.append(word)
    return out
```