2.2 - Control Flow

Ha Khanh Nguyen (hknguyen)

1. `if`, `elif`, and `else` Statements

1.1 Standard statements

- The `if` statement checks a condition that, if `True`, evaluates the code in the block that follows:

```
if x < 0:
  print('It\'s negative!')
```

- Note that a colon `:` is needed at the `if` statement, before the code block begins.
- And the code block after `:` MUST BE INDENTED.
  - Jupyter Notebook automatically indent the code after a colon `:`.
  - If you want to manually indent a line of code, use TAB! Please avoid using the spacebar.
- An `if` statement can be **optionally** followed by one or more `elif` blocks.
- The `else` statement is used a catch-all block if all of the conditions above it are `False`.
- If any of the condition is `True`, no further `elif` or `else` blocks will be reached.

```
if x < 0:
  print('It\'s a negative')
elif x == 0:
  print('Equal to zero')
elif 0 < x < 5:
  print('Positive but smaller than 5')
else:
  print('Positive and larger than or equal to 5')
```

Exercise 1: We are given 2 numbers stored in variables `a` and `b`. Write a program to print the number with the larger value.

Exercise 2: We are given 2 strings `s1` and `s2`. If one is contained in the other, print "One is a substring of the other!". Otherwise, print "They are distinct strings!"

1.2 Ternary expressions

- A *ternary expression* in Python allows you to combine an `if-else` block that produces a value into a single line or expression. The syntax for this in Python is:

```
value = true-expr if condition else false-expr
```

- The above code segment is equivalent to

```
if condition:
  value = true-expr
else:
  value = false-expr
```

- An example of ternary expression:

```
total = 0
x = 3
total = total + x if x % 2 == 0 else total
```

2. Loops

2.1 `for` loops

- `for` loops are for iterating over a collection (like a list or tuple) or an iterater.
- The standard syntax for a `for` loop is:

```
for value in collection:
  # do something with value
```

- An example of `for` loop with string:

```
s = 'hello'
for c in s:
  print(c)
```

```
## h
## e
## l
## l
## o
```

```
s = 'hello'
for i in range(len(s)):
  print(s[i])
```

```
## h
## e
## l
## l
## o
```

- An example of `for` loop with list:

```
nums = [11, 2, 8, 4, 5]
sum = 0
for i in nums:
  sum = sum + i
print(sum)
```

```
## 30
```

```
nums = [11, 2, 8, 4, 5]
sum = 0
for i in range(len(nums)):
  sum = sum + nums[i]
print(sum)
```

```
## 30
```

2.2 `range()` function

- The `range()` function returns an iterator that yields a sequence of evenly spaced integers.
  - An iterator ≈ an object that can be iterated.

```
range(10)
```

```
## range(0, 10)
```

```
list(range(10))
```

```
## [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

- `range(start, end, step)`:

```
list(range(0, 20, 2))
```

```
## [0, 2, 4, 6, 8, 10, 12, 14, 16, 18]
```

```
list(range(5, 0, -1))
```

```
## [5, 4, 3, 2, 1]
```

2.3 `continue` keyword

- You can advance a `for` loop to the next iteration, skipping the remainder of the block, using the `continue` keyword.
- Consider this code, which sums up integers in a list and skips `None` values:

```
sequence = [1, 2, None, 4, None, 5]
total = 0
for value in sequence:
  if value is None:
    continue
  total += value
print(total)
```

```
## 12
```

2.4 `break` keyword

- A `for` loop can be exited altogether with the `break` keyword.
- This code sums elements of the list until a 5 is reached:

```
sequence = [1, 2, 0, 4, 6, 5, 2, 1]
total_until_5 = 0
for value in sequence:
  if value == 5:
    break
  total_until_5 += value
print(total_until_5)
```

```
## 13
```

- The `break` keyword only terminates the innermost for loop; any outer for loops will continue to run:

```
for i in range(4):
  for j in range(4):
    if j > i:
      break
    print((i, j))
```

```
## (0, 0)
## (1, 0)
## (1, 1)
## (2, 0)
## (2, 1)
## (2, 2)
## (3, 0)
## (3, 1)
## (3, 2)
## (3, 3)
```

2.5 `while` loops

- A `while` loop specifies a condition and a block of code that is to be executed until the condition evaluates to `False` or the loop is explicitly ended with `break`.

```
x = 256
total = 0
while x > 0:
  if total > 500:
    break
  total += x
  x = x // 2
print(total)
```

```
## 504
```