Intermediate Python

Ed Davis

25/03/2021

Intermediate Python

(Note that a python terminal can be started in RStudio by typing `reticulate::repl_python()`, and it can be quit using the `exit` command)

String formatting

Variables and strings can be output in python using the `print()` function.

```
print("Hello world")
```

```
## Hello world
```

```
hello_var = "Hello World"
print(hello_var)
```

```
## Hello World
```

```
pi = 3.14159
print(pi)
```

```
## 3.14159
```

Print statements can include both strings and variables either by separating strings and variables with `,`. To embed a variable within a string, one could use either the "old style" formatting using the `%` operator, the "new style" string formatting (`str.format`) or "string interpolation" (as known as "f-strings") for python 3.6 and above.

```
print("Comma separated: The value of pi is ", pi)
```

```
## Comma separated: The value of pi is  3.14159
```

```
print("Old Style: The value of pi is %0.2f to 2 decimal places"%pi)
```

```
## Old Style: The value of pi is 3.14 to 2 decimal places
```

```
print("str.format: The value of pi is {:.2f} to 2 decimal places".format(pi))
```

```
## str.format: The value of pi is 3.14 to 2 decimal places
```

```
print(f"f-Strings: The value of pi is {pi:.2f} to 2 decimal places")
```

```
## f-Strings: The value of pi is 3.14 to 2 decimal places
```

List Comprehensions and loops

In base python a list can be defined using `list = []`. Common practice to fill the list involve using loops and `list.append(item)`. Loops encapsulate indented code after the `:` at the end of a loop declaration. When looping over a list, it is common to use `for i in range(len(list)):`. The `range` function defines iterations for `i` from 0 to the argument, in this case `len(list)`, which is the length of the list. This will iterate over all the items in `list`.

```
hello = "Hello World"
hello_letters = []
for i in range(len(hello)):
  hello_letters.append(hello[i])

hello_letters
```

```
## ['H', 'e', 'l', 'l', 'o', ' ', 'W', 'o', 'r', 'l', 'd']
```

Additionally, loops can define the iterator as items in the list, not just indices of the list.

```
hello = "Hello World"
hello_letters = []
for letter in hello:
  hello_letters.append(letter)

hello_letters
```

```
## ['H', 'e', 'l', 'l', 'o', ' ', 'W', 'o', 'r', 'l', 'd']
```

Functions

Functions in python are declared using `def function_name(args):` and the functions encompass everything that is indented after the `:`, similar to loops. An overview of the function can also be included using `"""doc strings"""`. Functions are not required to return anything but they can return variables using `return var`.

```
def Reverse(tuples):
    """Reverse a list of tuples"""
    new_tup = ()
    for k in reversed(tuples):
        new_tup = new_tup + (k,)
    return new_tup

Reverse([1,2,3,4,5])
```

```
## (5, 4, 3, 2, 1)
```

Classes and Objects

Classes are declared using `class foo:` and this generally encapsulates at least one class or member function. A class must have a constructor member function, called `__init__`. In python classes can self-reference through the use of `self`, which is used as an argument for each member function as these functions will, by definition, act on the class. Member data is also created and modified using the `self.member_data` syntax.

```
class person:
  """Class containing data about people"""

  def __init__(self, name, height, likes_maths):

    # Set initial member data
    self.name = name
    self.height = height
    self.likes_maths = likes_maths

  def summary(self):
    """Prints a string describing the person"""

    maths_opinion = ["cool" if self.likes_maths is True else "rubbish"][0]
    print(f"{self.name} has a height of {self.height} cm and thinks that maths is {maths_opinion}")

person_1 = person("Ed", 180, True)
person_1.summary()
```

```
## Ed has a height of 180 cm and thinks that maths is cool
```

Exceptions

Consider the code below. We see that a problem arises when a person gets stuck on maths for too long without taking a break, they don't like maths anymore.

```
class person:
    """Class containing data about people"""

    def __init__(self, name, height, likes_maths):

          # Set initial member data
          self.name = name
          self.height = height
          self.likes_maths = likes_maths
          self.maths_tolerance = 3

    def attempt_hard_problem(self):
        """Function call means that the person gets stuck with maths"""

        self.maths_tolerance -= 1
        if self.maths_tolerance <= 0:
            self.likes_maths = False

    def take_break(self):
        """Function call means that the person takes a break from a maths problem"""
```

```
        self.maths_tolerance += 1
        if self.maths_tolerance >= 0:
            self.likes_maths = True

    def summary(self):
        """Prints a string describing the person"""

        maths_opinion = ["cool" if self.likes_maths is True else "rubbish"][0]
        print(f"{self.name} has a height of {self.height} cm and thinks that maths is {maths_opinion}")


person_1 = person("Ed", 180, True)

person_1.attempt_hard_problem()
person_1.attempt_hard_problem()
person_1.attempt_hard_problem()
person_1.summary()
```

```
## Ed has a height of 180 cm and thinks that maths is rubbish
```

We can prevent this by raising an error using `raise RuntimeError("error string")`. This stops the code and displays the error message.

```
class person:
    """Class containing data about people"""

    def __init__(self, name, height, likes_maths):

          # Set initial member data
          self.name = name
          self.height = height
          self.likes_maths = likes_maths
          self.maths_tolerance = 3

    def attempt_hard_problem(self):
        """Function call means that the person gets stuck with maths"""

        self.maths_tolerance -= 1
        if self.maths_tolerance <= 0:
            raise RuntimeError(f"{self.name} is being overworked.")
            self.likes_maths = False

    def take_break(self):
        """Function call means that the person takes a break from a maths problem"""

        self.maths_tolerance += 1
        if self.maths_tolerance >= 0:
            self.likes_maths = True

    def summary(self):
        """Prints a string describing the person"""
```

```
        maths_opinion = ["cool" if self.likes_maths is True else "rubbish"][0]
        print(f"{self.name} has a height of {self.height} cm and thinks that maths is {maths_opinion}")


person_1 = person("Ed", 180, True)

person_1.attempt_hard_problem()
person_1.attempt_hard_problem()
person_1.attempt_hard_problem()
```

```
## Error in py_call_impl(callable, dots$args, dots$keywords): RuntimeError: Ed is being overworked.
##
## Detailed traceback:
##   File "<string>", line 1, in <module>
##   File "<string>", line 17, in attempt_hard_problem
```

While this prevents the person from disliking maths, it also breaks the simulation. This can be solved through the use of `try` and `except`. If the code encapsulated by `try` fails to run, the program skips to the code encapsulated by `except`. In the case where the `try` code works, the `except` code is not run. We can therefore stop the person from disliking maths by by using `try` to attempt the hard problem, but if the person is being overworked, it skips to `except` which tells the users to let the person take a break.

```
class person:
    """Class containing data about people"""

    def __init__(self, name, height, likes_maths):

          # Set initial member data
          self.name = name
          self.height = height
          self.likes_maths = likes_maths
          self.maths_tolerance = 3

    def attempt_hard_problem(self):
        """Function call means that the person gets stuck with maths"""

        self.maths_tolerance -= 1
        if self.maths_tolerance <= 0:
            raise RuntimeError(f"{self.name} is being overworked.")
            self.likes_maths = False

    def take_break(self):
        """Function call means that the person takes a break from a maths problem"""

        self.maths_tolerance += 1
        if self.maths_tolerance >= 0:
            self.likes_maths = True

    def summary(self):
        """Prints a string describing the person"""

        maths_opinion = ["cool" if self.likes_maths is True else "rubbish"][0]
        print(f"{self.name} has a height of {self.height} cm and thinks that maths is {maths_opinion}")
```

```python

person_1 = person("Ed", 180, True)

try:
    person_1.attempt_hard_problem()
    person_1.attempt_hard_problem()
    person_1.attempt_hard_problem()
except RuntimeError:
    print(f"{person_1.name} should probably take a break")
```

```
## Ed should probably take a break
```