Functional Programming 2

Contents

- First-class functions
- Pure functions
- Immutable Data
- Strict and non-strict evaluation - eager vs. lazy evaluation

First-class functions

- Functional programming is often succinct and expressive.
- Can achieve this is by providing functions as arguments and return values for other functions.
- Functions must be first-class objects in the runtime environment.
  - A **first class object** is an entity that can be dynamically created, destroyed, passed to a function, returned as a value, and have all the rights as other variables in the programming language have.
- Functions are objects that are created by def statement.
- We can also create a function
  - as a callable object
  - by assigning lambda to a variable.

Pure Functions

- A pure function does not produce side effects.
- It communicates with the calling program only through parameters (which it does not modify) and a return value.

Pure Functions

- Here is the doubleStuff function written as a pure function.
- To use the pure function version of double_stuff to modify things, we would assign the return value back to things.

```
1  def doubleStuff(a_list):
2      """ Return a new list in which contains doubles of the elements in a_list. """
3      new_list = []
4      for value in a_list:
5          new_elem = 2 * value
6          new_list.append(new_elem)
7      return new_list
8
9  things = [2, 5, 9]
10 print(things)
11 things = doubleStuff(things)
12 print(things)
13
```

```
[2, 5, 9]
[4, 10, 18]
```

Pure Functions

- A function is called pure function if it always returns the same result for same argument values and it has no side effects like modifying an argument (or global variable) or outputting something.
- The only result of calling a pure function is the return value.
- Examples of pure functions are strlen(), pow(), sqrt() etc. Examples of impure functions are printf(), rand(), time(), etc.

Pure Functions Benefits

- The benefit of using pure functions over impure (non-pure) functions is the reduction of side effects.
- Programmers reduce side effects in their code to make it easier to follow, test, and debug.
- The more side effects a codebase has, the harder it is to step through a program and understand its sequence of execution.

Pure Functions

- Write local-only code.
- Avoid the global statements.
- No use of nonlocal, it is a side effect in another scope.
- A Python lambda is a pure function.

```
>>>func = lambda x: 2**x-1
>>>func(17)
131071
```

- Lambda function can't have assignment statements.
- They are always pure functions and suitable for functional programming.

Difference Referential Transparency vs Pure Expression

- Referential Transparency :
  - If all functions involved in the expression are pure functions, then the expression is referentially transparent.
  - Some impure functions can be included in the expression if their values are discarded and their side effects are insignificant.
- Pure Expression:
  - Pure functions are required to construct pure expressions.
  - Pure expressions are often referred to as being referentially transparent.

Immutable Data

- Make extensive use of immutable data structures-tuples.
- Avoid class definitions
- Functional programming doesn't need stateful objects.

Strict and non-strict evaluation

- The idea of lazy or non-strict evaluation is very helpful.
- logical expression operators and, or, and if-then-else are all non-strict.
- *short-circuit* operators - don't need to evaluate all arguments.
- strict – other than logical operators, an expression is evaluated eagerly from left-to-right.
- A sequence of statement lines is also evaluated strictly in order.
- Literal lists and tuples require eager evaluation.

Strict and non-strict evaluation

- When a class is created, the method functions are defined in a strict order.
  - If we provide two methods with the same name, the second one is retained because of the strict evaluation order.
- Python's generator expressions and generator functions are lazy.