Introduction to Programming - Part 2 Solutions

Loops

1.

`py for i in range(11): print(i)`

1.

`py i = 0 while i <= 10: print(i) i += 1`

1.

```py nums = [0, 2, 8, 20, 43, 82, 195, 204, 367]

for num in nums: print(num) ```

1.

`py for i in range(11): print(i) else: print('Done!')`

1.

```py list1 = ["apple", "banana", "cherry", "durian", "elderberry", "fig"] list2 = ["avocado", "banana", "coconut", "date", "elderberry", "fig"]

for x in list1: for y in list2: if x == y: print(x) ```

1.

```py while True: x = random.randint(1, 100) if x % 5 == 0: print('multiple of 5: stopping loop') break

```
elif x % 3 == 0:
  print('multiple of 3: skipping iteration')
  continue

else:
  print(x)
```

```

Dictionaries

1.

```py car = { 'brand': 'Ford', 'model': 'Mustang', 'year' : 1964, 'isNew': False }

car['colour'] = 'red' print(car['colour']) # red ```

1.

```py car = { 'brand': 'Ford', 'model': 'Mustang', 'year' : 1964, 'isNew': False }

car['model'] = 'fiesta' print(car['model']) # fiesta ```

1.

```py car = { 'brand': 'Ford', 'model': 'Mustang', 'year' : 1964, 'isNew': False }

del car['model'] print(car) # {'brand': 'Ford', 'year': 1964, 'isNew': False} ```

1.

```py car = { 'brand': 'Ford', 'model': 'Mustang', 'year' : 1964, 'isNew': False }

for key, value in car.items(): print("key: " + key + ", value: " + str(value))

# key: brand, value: Ford # key: model, value: Mustang # key: year, value: 1964 # key: isNew, value: False ```

## Functions

1.

```py def add_numbers(a, b): return a + b

result = add_numbers(1, 2) print(result) # 3 ```

1.

```py def add_numbers(*nums): result = 0 for num in nums: result += num

```
  return result
```

result = add_numbers(1, 2, 3, 4) print(result) # 10 ```

1.

```py def create_dictionary(**kwargs): dictionary = {}

```
  for key, value in kwargs.items():
    dictionary[key] = value

  return dictionary
```

dic = create_dictionary(title="The Matrix", director="Wachowski", year=1999) print(dic) # {'title': 'The Matrix', 'director': 'Wachowski', 'year': 1999} ```

## Fib Solution
```

```
# ============================
# Solution 1
# ============================


def fib(n: int) -> int:

    if n <= 2:
        return 1

    return fib(n-1) + fib(n-2)


print(fib(1))  # 1
print(fib(5))  # 5
print(fib(7))  # 13

# This is great but this takes forever:
print(fib(50))

# The recursion tree looks like this:
#
# fib(5) -> fib(4) -> fib(3) -> fib(2) -> 1
#                            -> fib(1) -> 1
#                  -> fib(2) -> 1
#        -> fib(3) -> fib(2) -> 1
#                  -> fib(1) -> 1
#
# Add up all the `1s` and we get the answer: `fib(5) == 5`
# Look at how many times we worked out `fib(3)`, and `fib(2)` again... what a waste.
# Imagine how complicated this is for `fib(50)` and now we see why it's slow.




# ============================
# Solution 2
# ============================

# Let's add memoisation so that before we work out fib(n), we check if we've already done the work and if so return the result inste


def fast_fib(n: int, memo: dict = {}) -> int:

    if n <= 2:
        return 1

    if n in memo:
        return memo[n]

    memo[n] = fast_fib(n-1, memo) + fast_fib(n-2, memo)

    return memo[n]


# That was fast:
print(fast_fib(100))

# So was this:
print(fast_fib(1000)) # Will have to extend max recursion dept to go beyond this
```