Module 2. Functions, loops and conditionals.

Functions.

Define functions:

```
def double(x):
  doubled_x = x * 2
  return doubled_x
```

1. Define.
    a. On the first line we start defining a function, hence that line starts with the keyword def.
    We name this function double
    b. The parameter takes a single input variable that we name x
    c. Note that the parameter goes between parentheses and we add a colon afterwards
2. Describe
    a. On the second line we start writing what the function should do.
    We use the input variable x as though we declared the variable ourselves previously.

    b. We write doubled_x = x * 2, which is nothing special in itself.
    c. The third line states: return doubled_x. This concludes the function and returns doubled_x.

Reuse functions:

Simply call the function and add and argument

```
double(4)
#will return 8
```

Multiple parameters:

You can use several parameters within a function

```
def multiply(a, b):
  return a * b

two_and_two = multiply(2, 2)
print(two_and_two)
```

Keyword arguments:

It's very important to follow the argument order described by the function definition. To avoid using the same order, you can use the keyword argument:

```
def greet_user(name, birth_year):
     Age = 2023 - birth_year
     print(f"Hi {name}! You are {age} years old!.")

greet_user(name="John", birth_year=1986)
```

Default parameters:

Include the parameter and the argument when defining the function. That will create a default argument.

```


def weather_report(day="today")
     print(f'Here is the weather for {day}')
     #get weather info…
     return info

weather_report()
```

In the case of several parameters, required parameters go first!

```
def weather_report(city, day="today")
     print(f'Here is the weather for {day}')
     #get weather info…
     return info

weather_report('Amsterdam')
```

Conditionals

IF and Else

Add the condition inside a function. The logic is: "if this happens: then do this. Else: do that". Mind the colons at the end of the statements.

```
def eligible_to_vote(age, nationality):
     if age >= 18:
          return True
     else:
          return False
```

You can use operators to add more conditions

```
def eligible_to_vote(age, nationality):
     if age >= 18 and nationality == 'Italian':
          return True
     else:
          return False
```

ELIF

Elif stands for Else if and adds a new condition. Don't confuse it with an and. In no code language is something like: if this happens, do this, but if it doesn't and this other thing happens, then to this.

```
def eligible_to_vote(age, nationality):
     if age >= 18 and nationality == ‘Italian’:
          return True
     elif age >=25 and nationality == ‘Portuguese’:
          return True
     else:
          return False
```

Data collections

Lists

A list is an item just like strings, integers and float. They can hold a mix of items, even other lists!
Lists are mutable.

Declare the variable and and the values between brackets, separated by a comma.

```
names = [‘Kelso’, ‘Cox’, ‘Turk’, ‘Reid’]
```

You can operate with lists: index, slice, compare, order, cast…
Check the example REPLit here.

Sets and Tuples

Sets are "unordered lists", defined using curly braces {} or the set() function.

- Are unordered
- Unindexed
- Immutable

```
#Option A
names = [‘Kelso’, ‘Cox’, ‘Turk’, ‘Reid’]
set(names)

#option B
names_set = {‘Kelso’, ‘Cox’, ‘Turk’, ‘Reid’}
```

Tuples are similar to lists, but they are immutable, defined using parentheses (). They are useful in situations where we need to represent fixed collections of data that should not be modified.

```
alice = (‘Alice’, 5)
```

Dictionaries

A dictionary (dict) is defined using curly braces { } and consists of key-value pairs, where each key is associated with a value using a colon (:) between them.

They are mutable, ordered and do not use indexes, but keys.

```
Product = {
      "name"  : "tofu",
      "Price": "2"
}
```

You can nest data collections and access data within. Check the REPLit link here.

Summary

Lists	Sets	Tuples	Dictionaries
Ordered	Unordered	Ordered	Ordered
Mutable (changeable)	Mutable (but its items are not)	Immutable (unchangeable)	Mutable
Allows duplicates	Does not allow duplicates	Allows duplicates	Does not allow duplicates
Indexed	Unindexed	Indexed	Indexed
List items enclosed within []	Setitems enclosed within {}	Tuple items are enclosed within ()	Dictionary items enclosed within {}

Loops

The For loop

It iterates over a sequence of elements or a specified range, a programming construct that allows you to repeat a block of code for a specific number of times or iterate over a sequence of elements.

```
names = [‘Kelso’, ‘Cox’, ‘Turk’, ‘Reid’]
for name in names:
      print(f’Hello Dr. {name}!’}
```

Notice that ‘name’ is a local variable that only exists in the scope of the For loop.

range()
The range() function produces a range of numbers based on the parameters we provide to it. The output type is iterable, so we can use it directly in a for loop.

Break and continue:

Using these keywords is not a good practice

When the **break** is reached, the loop that the break is in will immediately stop and the code execution continues after the loop. In other words, we stop the loop before it would loop through all the items.
print('We have a long road ahead.')

```
for i in range(100):
    print(i)
    if i == 10:
        break
    print('Almost there...')
print("That wasn't so bad after all.")
```

The other expression that we can use to control the loop is the continue expression.

When using **continue**, instead of breaking out of the loop completely it only skips to the next iteration.

```
print('We have a long road ahead.')
for i in range(10):
  if i % 2 == 0:
    print(i)
    continue
  print('Almost there...')
print("That wasn't that bad either.")
```

While loop

It keeps looping until the specified condition is no longer true, or a break-statement has been reached.

```
i = 10
while i > 0:
  print(i)
  i -= 1
```