Business Intelligence-relaterade programspråk

OO, Math

Lambda

- A lambda function is a small anonymous function.
- Use lambda functions when an anonymous function is required for a short period of time.
- A lambda function can take any number of arguments, but can only have one expression.

```
lambda arguments : expression
```

```
x = lambda a : a + 10
print(x(5))
```

- Multiply argument a with argument b and return the result:

```
x = lambda a, b : a * b
print(x(5, 6))
```

- Summarize argument a, b, and c and return the result:

```
x = lambda a, b, c : a + b + c
print(x(5, 6, 2))
```

Why Use Lambda Functions?

- The power of lambda is better shown when you use them as an anonymous function inside another function.
- Say you have a function definition that takes one argument, and that argument will be multiplied with an unknown number:

```
def myfunc(n):
  return lambda a : a * n
```

- Use that function definition to make a function that always doubles the number you send in:

```
def myfunc(n):
  return lambda a : a * n

mydoubler = myfunc(2)
print(mydoubler(11))
```

- Or, use the same function definition to make a function that always triples the number you send in:

```
def myfunc(n):
  return lambda a : a * n

mytripler = myfunc(3)
print(mytripler(11))
```

Why Use Lambda Functions?

- Or, use the same function definition to make both functions, in the same program:

```
def myfunc(n):
  return lambda a : a * n

mydoubler = myfunc(2)
mytripler = myfunc(3)

print(mydoubler(11))
print(mytripler(11))
```

Avancerade variabler

- Saker i riktiga världen består ofta att olika variabler.
- Hänger de ihop på något sätt?

```
name = "Micke";
age = 44;
shoe_size = 43;
```

Objekt & klasser

- Ett objekt är i sin enklaste form en variabel som har under-variabler.
- En klass är lite som en ritning eller mall för ett objekt.

Classes and Objects

- Python is an object oriented programming language.
- Almost everything in Python is an object, with its properties and methods.
- A Class is like an object constructor, or a "blueprint" for creating objects.

Create a Class and an Object

- Create a class named MyClass, with a property named x:

```
class MyClass:
  x = 5
```

- Klassen kan nu användas som mall för objekt.
- Create an object named p1, and print the value of x:

```
p1 = MyClass()
print(p1.x)   # 5
```

The __init__() Function

- The examples above are classes and objects in their simplest form, and are not really useful in real life applications.
- To understand the meaning of classes we have to understand the built-in `__init__()` function.
- All classes have a function called `__init__()`, which is always executed when the class is being initiated.
- Use the `__init__()` function to assign values to object properties, or other operations that are necessary to do when the object is being created.

The __init__() Function

- `__init__()` is **not** a constructor.
- `__init__()` is called immediately after the object is created and is used to initialize it.
- The constructor is called `__new__ ()`.
- https://medium.com/analytics-vidhya/is-init-the-constructor-in-python-43bb744e73b

The __init__() Function

- Create a class named Person, use the `__init__()` function to assign values for name and age:

```
class Person:
  def __init__(self, name, age):
    self.name = name
    self.age = age

p1 = Person("John", 36)

print(p1.name)
print(p1.age)
```

Exercise

- Create a class named Vehicle, use the `__init__()` function to assign values for make, model, color, year, velocity and no_of_wheels.
- Velocity should always be set to 0.
- Let no_of_wheels be optional when creating an object by setting the default value 4 for the appropriate `__init__()` parameter.
- Create a couple of vehicles and print out their properties.

Inheritance

- Inheritance allows us to define a class that inherits all the methods and properties from another class.
- *Parent class* is the class being inherited from, also called *base class* or *super class*.
- *Child class* is the class that inherits from another class, also called *derived class* or *sub class*.

Create a Parent Class

- Any class can be a parent class, so the syntax is the same as creating any other class.
- Create a class named Person, with `firstname` and `lastname` properties, and a `printname` method:

```
class Person:
  def __init__(self, fname, lname):
    self.firstname = fname
    self.lastname = lname

  def printname(self):
    print(self.firstname, self.lastname)

# Use the Person class to create an object, and then execute the printname method:

x = Person("John", "Doe")
x.printname()
```

Create a Child Class

- To create a class that inherits the functionality from another class, send the parent class as a parameter when creating the child class.
- Create a class named Student, which will inherit the properties and methods from the Person class.

```
class Student(Person):
  pass
```

- *Note:* Use the pass keyword when you do not want to add any other properties or methods to the class.

- Now the Student class has the same properties and methods as the Person class.
- Use the Student class to create an object, and then execute the `printname` method:

```
x = Student("Mike", "Olsen")
x.printname()
```

Add the __init__() Function

- So far we have created a child class that inherits the properties and methods from its parent.
- We want to add the `__init__()` function to the child class (instead of the pass keyword).
- *Note:* The `__init__()` function is called automatically every time the class is being used to create a new object.

- Add the `__init__()` function to the Student class.

```
class Student(Person):
  def __init__(self, fname, lname):
    #add properties etc.
```

- When you add the `__init__()` function, the child class will no longer inherit the parent's `__init__()` function.
- Note: The child's `__init__()` function overrides the inheritance of the parent's `__init__()` function.

Add the __init__() Function

- To keep the inheritance of the parent's `__init__()` function, add a call to the parent's `__init__()` function.

```
class Student(Person):
  def __init__(self, fname, lname):
    Person.__init__(self, fname, lname)
```

- Now we have successfully added the `__init__()` function, and kept the inheritance of the parent class, and we are ready to add functionality in the `__init__()` function.

Use the super() Function

- Python also has a `super()` function that will make the child class inherit all the methods and properties from its parent.

```
def __init__(self, fname, lname):
  super().__init__(fname, lname)
```

- By using the `super()` function, you do not have to use the name of the parent element, it will automatically inherit the methods and properties from its parent.

Add Properties

- Add a property called `graduationyear` to the Student class.

```
class Student(Person):
  def __init__(self, fname, lname):
    super().__init__(fname, lname)
    self.graduationyear = 2019
```

- In the example below, the year 2019 should be a variable, and passed into the Student class when creating student objects. To do so, add another parameter in the `__init__()` function:

```
class Student(Person):
  def __init__(self, fname, lname, year):
    super().__init__(fname, lname)
    self.graduationyear = year

x = Student("Mike", "Olsen", 2019)
```

Methods

- Objects have properties, ”sub variables”.
- Objects can also have functions.
  - Functions in classes are called *methods*.

Add Methods

- Add a method called welcome to the Student class.

```
class Student(Person):
  def __init__(self, fname, lname, year):
    super().__init__(fname, lname)
    self.graduationyear = year

  def welcome(self):
    print("Welcome", self.firstname, self.lastname, "to the class of",
self.graduationyear)
```

- If you add a method in the child class with the same name as a function in the parent class, the inheritance of the parent method will be overridden.

Exercise

- Add the max_speed property to the Vehicle class.
- Create the methods `accelerate()` and `decelerate()` that both take a change parameter and increases / decreases the velocity.
  - Check the parameter. The new speed can’t be lower than 0 or higher than the max speed.
- Create a class named Car and let it inherit from Vehicle. The max_speed should always be 160.
- Create a class named Truck and let it inherit from Vehicle.
  - The max_speed should always be 80.
  - Add the property load_capacity.
- Create a couple of cars and trucks and print out their properties.
- Create the method print() that prints out the properties for the vehicle.
- Create the class Motorcycle that inherits from Vehicle. Max speed is 200.

Iterators

- An iterator is an object that contains a countable number of values.
- An iterator is an object that can be iterated upon, meaning that you can traverse through all the values.
- Technically, in Python, an iterator is an object which implements the iterator protocol, which consist of the methods `__iter__()` and `__next__()`.

Iterator vs Iterable

- Lists, tuples, dictionaries, and sets are all iterable objects. They are iterable containers which you can get an iterator from.
- All these objects have a `iter()` method which is used to get an *iterator*.
- Return an iterator from a tuple, and print each value:

```
mytuple = ("apple", "banana", "cherry")
myit = iter(mytuple)

print(next(myit))
print(next(myit))
print(next(myit))
```

Iterator vs Iterable

- Even strings are iterable objects, and can return an iterator.
- Strings are also iterable objects, containing a sequence of characters.

```
mystr = "banana"
myit = iter(mystr)

print(next(myit))
print(next(myit))
print(next(myit))
print(next(myit))
print(next(myit))
print(next(myit))
print(next(myit))
```

Looping Through an Iterator

- We can use a `for` loop to iterate through an iterable object.
- Iterate the values of a tuple:

```
mytuple = ("apple", "banana", "cherry")

for x in mytuple:
  print(x)
```

- The `for` loop actually creates an iterator object and executes the `next()` method for each loop.

Create an Iterator

- To create an object/class as an iterator you have to implement the methods `__iter__()` and `__next__()` to your object.
- As you have learned in the Python Classes/Objects chapter, all classes have a function called `__init__()`, which allows you to do some initializing when the object is being created.
- The `__iter__()` method acts similar, you can do operations (initializing etc.), but must always return the iterator object itself.
- The `__next__()` method also allows you to do operations, and must return the next item in the sequence.

Create an Iterator

- Create an iterator that returns numbers, starting with 1, and each sequence will increase by one (returning 1,2,3,4,5 etc.)

```
class MyNumbers:
  def __iter__(self):
    self.a = 1
    return self

  def __next__(self):
    x = self.a
    self.a += 1
    return x

myclass = MyNumbers()
myiter = iter(myclass)

print(next(myiter))
print(next(myiter))
print(next(myiter))
print(next(myiter))
print(next(myiter))
```

StopIteration

- The example above would continue forever if you had enough `next()` statements, or if it was used in a `for` loop.
- To prevent the iteration to go on forever, we can use the `StopIteration` statement.
- In the `__next__()` method, we can add a terminating condition to raise an error if the iteration is done a specified number of times

```
class MyNumbers:
  def __iter__(self):
    self.a = 1
    return self

  def __next__(self):
    if self.a <= 20:
      x = self.a
      self.a += 1
      return x
    else:
      raise StopIteration

myclass = MyNumbers()
myiter = iter(myclass)

for x in myiter:
  print(x)
```

Exercise

- Write an iterator class `reverse_iter`, that takes a list and iterates it from the reverse direction.

```
it = reverse_iter([1, 2, 3, 4])

next(it)   # 4
next(it)   # 3
next(it)   # 2
next(it)   # 1
next(it)   # StopIteration
```

Generators

- Generators simplifies creation of iterators.
- A generator is a function that produces a sequence of results instead of a single value.

```
def yrange(n):
    i = 0
    while i < n:
        yield i
        i += 1
```

- Each time the yield statement is executed the function generates a new value.

```
y = yrange(3)
next(y)   # 0
next(y)   # 1
next(y)   # 2
next(y)   # StopIteration
```

Generators

- So a generator is also an iterator. You don’t have to worry about the iterator protocol.
- The word “generator” is confusingly used to mean both the function that generates and what it generates. In this chapter, I’ll use the word “generator” to mean the genearted object and “generator function” to mean the function that generates it.
- When a generator function is called, it returns a generator object without even beginning execution of the function. When next method is called for the first time, the function starts executing until it reaches yield statement. The yielded value is returned by the next call.

Generators

- The following example demonstrates the interplay between `yield` and call to `__next__` method on generator object.

```
def integers():
    """Infinite sequence of integers."""
    i = 1
    while True:
        yield i
        i = i + 1
```

```
def squares():
    for i in integers():
        yield i * i

def take(n, seq):
    """Returns first n values from the given
sequence."""
    seq = iter(seq)
    result = []
    try:
        for i in range(n):
            result.append(next(seq))
    except StopIteration:
        pass
    return result

print(take(5, squares())) # prints [1, 4, 9, 16,
25]
```

Modules

- What is a Module?
  - Consider a module to be the same as a code library.
  - A file containing a set of functions you want to include in your application.

Create a Module

- To create a module just save the code you want in a file with the file extension `.py`.

```
def greeting(name):
  print("Hello, " + name)
```

Use a Module

- Now we can use the module we just created, by using the import statement.

```
import mymodule
mymodule.greeting(”Jonathan”)
```

- Note: When using a function from a module, use the syntax: `module_name.function_name`.

Naming a Module

- You can name the module file whatever you like, but it must have the file extension `.py`
- Re-naming a Module
  - You can create an alias when you import a module, by using the `as` keyword:

```
import mymodule as mx

a = mx.person1["age"]
print(a)
```

Built-in Modules

- There are several built-in modules in Python, which you can import whenever you like.
- Import and use the platform module

```
import platform

x = platform.system()
print(x)
```

Using the dir() Function

- There is a built-in function to list all the function names (or variable names) in a module, the `dir()` function.
- List all the defined names belonging to the platform module:

```
import platform

x = dir(platform)
print(x)
```

- *Note:* The `dir()` function can be used on all modules, also the ones you create yourself.

Import From Module

- You can choose to import only parts from a module, by using the `from` keyword.
- The module named `mymodule` has one function and one dictionary:

```
def greeting(name):
  print("Hello, " + name)

person1 = {
  "name": "John",
  "age": 36,
  "country": "Norway"
}
```

- Import only the person1 dictionary from the module:

```
from mymodule import person1

print (person1["age"])
```

- Note: When importing using the from keyword, do not use the module name when referring to elements in the module. Example: `person1["age"]`, **not** ~~`mymodule.person1["age"]`~~

Datetime

- A date in Python is not a data type of its own, but we can import a module named `datetime` to work with dates as date objects.
- Import the `datetime` module and display the current date:

```
import datetime

x = datetime.datetime.now()
print(x)
```

Date Output

- When we execute the code from the example above the result will be:

```
2021-03-30 03:26:10.505818
```

- The date contains year, month, day, hour, minute, second, and microsecond.
- The `datetime` module has many methods to return information about the date object.

Date Output

- Return the year and name of weekday:

```
import datetime

x = datetime.datetime.now()

print(x.year)
print(x.strftime("%A"))
```

Creating Date Objects

- To create a date, we can use the `datetime()` class (constructor) of the `datetime` module.
- The `datetime()` class requires three parameters to create a date: year, month, day.

```
import datetime

x = datetime.datetime(2020, 5, 17)

print(x)
```

- The datetime() class also takes parameters for time and timezone (hour, minute, second, microsecond, tzone), but they are optional, and has a default value of 0, (None for timezone).

The strftime() Method

- The `datetime` object has a method for formatting date objects into readable strings.
- The method is called `strftime()`, and takes one parameter, `format`, to specify the format of the returned string:
- Display the name of the month:

```
import datetime

x = datetime.datetime(2018, 6, 1)

print(x.strftime("%B"))
```

Date Format Reference

Directive	Description	Example
%a	Weekday, short version	Wed
%A	Weekday, full version	Wednesday
%w	Weekday as a number 0-6, 0 is Sunday	3
%d	Day of month 01-31	31
%b	Month name, short version	Dec
%B	Month name, full version	December
%m	Month as a number 01-12	12
%y	Year, short version, without century	18
%Y	Year, full version	2018
%H	Hour 00-23	17
%I	Hour 00-12	05
%p	AM/PM	PM
%M	Minute 00-59	41
%S	Second 00-59	08
%f	Microsecond 000000-999999	548513
%z	UTC offset	+0100
%Z	Timezone	CST
%j	Day number of year 001-366	365
%U	Week number of year, Sunday as the first day of week, 00-53	52
%W	Week number of year, Monday as the first day of week, 00-53	52
%c	Local version of date and time	Mon Dec 31 17:41:0
%x	Local version of date	12/31/18
%X	Local version of time	17:41:00
%%	A % character	%
%G	ISO 8601 year	2018
%u	ISO 8601 weekday (1-7)	1
%V	ISO 8601 weeknumber (01-53)	01

Exercise

- Write a program that takes an input date/time from a user and outputs the following formats.
    1. Current date and time
    2. Current year
    3. Month of year
    4. Week number of the year
    5. Weekday of the week
    6. Day of year
    7. Day of the month
    8. Day of week

Math

- Python has a set of built-in math functions, including an extensive math module, that allows you to perform mathematical tasks on numbers.
- Some built in functions:
  - The `min()` and `max()` functions can be used to find the lowest or highest value in an iterable:
  - x = min(5, 10, 25)
  - y = max(5, 10, 25)

Math

- The `abs()` function returns the absolute (positive) value of the specified number:

```
x = abs(-7.25)
print(x)
```

- The `pow(x, y)` function returns the value of `x` to the power of `y`.
- Return the value of 4 to the power of 3 (same as 4 * 4 * 4):

```
x = pow(4, 3)
print(x)
```

The Math Module

- Python has also a built-in module called math, which extends the list of mathematical functions.
- When you have imported the math module, you can start using methods and constants of the module.
- The `math.sqrt()` method for example, returns the square root of a number:

```
import math

x = math.sqrt(64)

print(x)
```

- The math.pi constant, returns the value of PI (3.14...)

The Math Module

- The `math.ceil()` method rounds a number upwards to its nearest integer, and the `math.floor()` method rounds a number downwards to its nearest integer, and returns the result.

```
import math

x = math.ceil(1.4)
y = math.floor(1.4)

print(x) # returns 2
print(y) # returns 1
```

The Math Module

- Read more in the documentation.
- https://docs.python.org/3/library/math.html

The Math Module

- Read more in the documentation.
- https://docs.python.org/3/library/math.html

Exercise

- Write the class `Rectangle` with properties `x` and `y`.
  - Add methods `area()` and `circumference()`.
- Write the class `Square` that inherits from `Rectangle`.
- Write the class `Circle` that uses a the `pi` constant.
- Write the class `Triangle` that uses different math methods like `tan`, `sin` and `cos` etc to calculate area and circumference depending on what sides you have access to.

Python
Övningar

- https://www.w3schools.com/python/exercise.asp
- https://www.w3resource.com/python-exercises/