Τμήμα Ηλεκτρολόγων Μηχανικών
και Μηχανικών Υπολογιστών

Μάθημα: **Τεχνικές Προγραμματισμού Υπολογιστών. (Εργαστηριακό μάθημα)**
Καθηγητής : **Πεφάνης Ευάγγελος**

5) Εργαστηριακές σημειώσεις στην γλώσσα προγραμματισμού Java Script.

Αντικειμενοστραφής προγραμματισμός JavaScript

Ένα αντικείμενο στην καθημερινή μας ζωή είναι κάτι το οποίο έχει μια υλική υπόσταση, με κάποια συγκεκριμένα χαρακτηριστικά αλλά επίσης και κάποιες λειτουργίες. Ένα αυτοκίνητο για παράδειγμα ως χαρακτηριστικά έχει το μοντέλο του, το έτος παρασκευής του, την μέγιστη ταχύτητα του, τα κυβικά εκατοστά της μηχανής του, και ως λειτουργίες έχει την εκκίνηση του, την χρήση του κινητήρα, την αλλαγή ταχύτητας κ.α.
Κάπως έτσι χρησιμοποιούνται και τα αντικείμενα σε κάθε γλώσσα προγραμματισμού. Παρόμοια λογική ακολουθεί και η javascript. Τα αντικείμενα είναι μεταβλητές στις οποίες μέσα σε αυτές έχει οριστεί ότι είναι απαραίτητο για την λειτουργία του εκάστοτε αντικειμένου. Το συντακτικό για την δημιουργία αντικειμένου έχει ως εξής:

```
var όνομα_αντικειμένου = {
 ιδιότητα_1: τιμή,
 ιδιότητα_2: τιμή,
 λειτουργία_1: function(παράμετρος_1, παράμετρος_2){
  ...κώδικας που εκτελείται...
  return τιμή;
 },
 λειτουργία_2: function(){
  ...κώδικας που εκτελείται...
  return τιμή;
 }
}
```

1. Χρησιμοποιούμε την λέξη «var» για να ορίσουμε το αντικείμενο μας.
2. Ακολουθεί η ονομασία του τηρώντας τους κανόνες ονομασίας μεταβλητών (λατινικοί χαρακτήρες κλπ).
3. Ύστερα αρχικοποιούμε το αντικείμενο με «=» ίσον.
4. Στην συνέχεια ένα ζεύγος αγκύλων περικλείει τις ιδιότητες (άλλοτε και χαρακτηριστικά) και τις λειτουργίες του αντικειμένου μας.
5. Κάθε χαρακτηριστικό ή λειτουργία αναγράφεται με λατινικούς χαρακτήρες και ύστερα ακολουθούν άνω κάτω τελείες. Μετά τις τελείες ακολουθεί ένας ορισμός τιμής ή λειτουργίας
6. Συνήθως αναγράφονται πρώτα οι ιδιότητες, οπότε και τους δίνουμε τιμές. Κάθε ιδιότητα μετά από το πέρας της δήλωσης τους, τις χωρίζουμε με κόμμα έτσι ώστε να προχωρήσουμε στην επόμενη.
7. Όταν φθάσουμε σε λειτουργίες, ακολουθούμε την ίδια λογική ονομασίας αλλά αντί για κάποια τιμή αναγράφουμε την λέξη «function» ύστερα ένα ζεύγος παρενθέσεων – αν θέλουμε παραμέτρους τις ορίζουμε – και ύστερα ένα ζεύγος αγκύλων όπου θα πλαισιωθεί ο κώδικας της λειτουργίας – μεθόδου του αντικειμένου.
8. Τέλος κλείνοντας την αγκύλη μιας λειτουργίας, πάντα συμπεριλαμβάνουμε και κόμμα αν έχουμε σκοπό να προσθέσουμε και άλλα στοιχεία αλλιώς αν πρόκειται για το τελευταίο στοιχείο, τότε απλά κλείνουμε με αγκύλη.

Ας δούμε ένα παράδειγμα αντικειμένου με βάση την παραπάνω περιγραφή του αυτοκινήτου:

```
var car = {
 model: 10,
 maxspeed: 300,
 gear:0,
 cc:1000,
 started:false,
 startEngine: function(){
```

```
  this.started = true;
 },
 changeGear: function(g){
  this.gear=g;
 }
}
```

Ας δούμε την αντιστοίχηση:

- Ιδιότητα Μοντέλο: model
- Ιδιότητα Μέγιστη ταχύτητα: maxspeed
- Ιδιότητα Τρέχουσα ταχύτητα: gear
- Ιδιότητα Κυβικά εκατοστά: cc
- Ιδιότητα Ενεργό: started
- Λειτουργία Εκκίνηση κινητήρα: startEngine
- Λειτουργία Αλλαγή ταχύτητας: changeGear

Με όλο τον παραπάνω κώδικα έχουμε κατασκευάσει ένα αντικείμενο με όνομα car και ιδιότητες-λειτουργίες όλες τις παραπάνω. Πως όμως τις χρησιμοποιούμε στον κώδικα μας ; Πολύ απλά χρησιμοποιούμε το όνομα του αντικειμένου ως προσδιοριστή και ύστερα όποια ιδιότητα η λειτουργία θέλουμε χωρισμένη με τελεία από το όνομα του αντικειμένου, πχ:

Ιδιότητα-λειτουργία	Τρόπος κλήσης
model	**car.model**
maxspeed	**car.maxspeed**
startEngine	**car.startEngine()**
changeGear	**car.changeGear(4)**

```
<!DOCTYPE HTML>
<html>
 <head>
  <meta charset="utf-8">
  <title>Αυτοκίνητο</title>
  <script type='text/javascript'>
   var car = {
    model: 10,
    maxspeed: 300,
    gear:0,
    cc:1000,
    started:false,
    startEngine: function(){
     this.started = true;
    },
    changeGear: function(g){
     this.gear=g;
    }
   }
  </script>
 </head>
 <body>
  <input type='button' value='Πάτησε εδώ!' onclick='alert("Το αυτοκίνητο
είναι:\n Μοντέλο:"+car.model+"\n Μέγιστη ταχύτητα:"+car.maxspeed+"\n Κυβικά
εκατοστά:"+car.cc);'/>
 </body>
</html>
```

Αντικείμενα.

Η JavaScript είναι μια λειτουργική γλώσσα και για τον αντικειμενοστραφή προγραμματισμό χρησιμοποιεί τόσο αντικείμενα όσο και λειτουργίες, αλλά τα αντικείμενα συνήθως χρησιμοποιούνται ως δομή δεδομένων, παρόμοια με ένα λεξικό της Python .

Για να αρχικοποιήσετε ένα αντικείμενο, χρησιμοποιήστε τα { }:

```
var emptyObject = {};
var personObject = {
    firstName : "John",
    lastName : "Smith"
}
```

Αντιμετώπιση μελών

Τα μέλη αντικειμένων μπορούν να αντιμετωπιστούν χρησιμοποιώντας τους χειριστές { } παρενθέσεις (ονομάζονται brackets στα αγγλικά) ή, όμοια με τους πίνακες, αλλά όπως και πολλές άλλες αντικειμενοστραφείς γλώσσες, μπορεί να χρησιμοποιηθούν και οι χειριστές [ ] .

Για παράδειγμα, μπορούμε να συνεχίσουμε να γεμίζουμε το αντικείμενο του ατόμου με περισσότερες λεπτομέρειες:

```
var personObject = {
    firstName : "John",
    lastName : "Smith"
}
personObject.age = 23;
personObject["salary"] = 14000;
```

Σημειώστε ότι οι μέθοδοι αντικειμένων στην JavaScript έχουν μια σταθερή σειρά, όπως οι πίνακες.

*Όπως ξέρουμε οι μεταβλητές στη Javascript είναι **δοχεία που περιέχουν μία τιμή**. Τα αντικείμενα στη Javascript είναι και αυτά μεταβλητές αλλά **μπορούν να περιέχουν πολλές τιμές**.*

Μπορούμε να δημιουργήσουμε και να δηλώσουμε ένα αντικείμενο με δύο τρόπους.

Πρώτος τρόπος:

```
var car = { type: "Toyota", model:"Yaris", color:"silver", horsePower: 85 };
```

Δεύτερος τρόπος:

```
var car = {
```

```
  type: "Toyota",
  model:"Yaris",
  color:"silver",
  horsePower: 85
};
```

*Τα αντικείμενα στη Javascript είναι σαν τα πραγματικά αντικείμενα. Έχουν **ιδιότητες και μεθόδους**.*
*Για παράδειγμα, ένα αυτοκίνητο είναι ένα αντικείμενο. Έχει **ιδιότητες**, όπως βάρος και χρώμα, και **μεθόδους**, όπως η εκκίνηση και η ακινητοποίηση του.*
Ιδιότητες (Properties) Αντικειμένων στη Javascript

*Όπως είδαμε παραπάνω τα αντικείμενα είναι μεταβλητές που μπορούν να περιέχουν πολλές τιμές. **Οι τιμές** αυτές **αποθηκεύονται στις ιδιότητες του αντικειμένου**.*
Στο παρακάτω παράδειγμα θα δημιουργήσουμε ένα άνθρωπο και θα του ορίσουμε τις εξής ιδιότητες: όνομα, επώνυμο, ηλικία, βάρος, χρώμα ματιών.

```
var person = { firstName: "Θανάσης", lastName: "Παπαδόπουλος", age: 50, weight: 85,
eyeColor: "brown" };
```

Μπορούμε να χρησιμοποιήσουμε τις ιδιότητες ενός αντικειμένου με δύο τρόπους:

Πρώτος τρόπος (ενδείκνυται)

```
person.firstName;

person.age;
```

Δεύτερος τρόπος:

```
person["firstName"];

person["age"];
```

Όλα τα αντικείμενα έχουν τις ίδιες ιδιότητες αλλά οι τιμές των ιδιοτήτων αλλάζουν από αντικείμενο σε αντικείμενο.

Μέθοδοι Αντικειμένων στη Javascript

*Τα αντικείμενα στη Javascript εκτός από τις τιμές, μπορούν να **περιέχουν και μεθόδους**. Ενέργειες δηλαδή που εκτελεί το αντικείμενο μας.*
Οι μέθοδοι των αντικειμένων είναι οι γνωστές συναρτήσεις (functions).

Στο παρακάτω παράδειγμα θα προσθέσουμε στον άνθρωπο που δημιουργήσαμε στο προηγούμενο παράδειγμα, την μέθοδο fullName που επιστρέφει το όνομα και το επώνυμο του.

```
var person = {
  firstName: "Θανάσης",
  lastName: "Παπαδόπουλος",
  age: 50,
  weight: 85,
  eyeColor: "brown"
  fullName : function() {
    return this.firstName + " " + this.lastName;
  }
};
```

Ο σωστός τρόπος να καλέσουμε μία μέθοδο ενός αντικειμένου είναι ο εξής:

```
name = person.fullName();
```

Αν καλέσουμε τη μέθοδο ενός αντικειμένου και ξεχάσουμε να γράψουμε τις παρενθέσεις, δηλαδή:

```
name = person.fullName;
```

Αυτό που θα μας επιστραφεί είναι ο ορισμός της μεθόδου.

```
function () { return this.firstName + " " + this.lastName; }
```

Η JavaScript λέξη-κλειδί this αναφέρεται στο αντικείμενο στο οποίο ανήκει.

JS Classes

ECMAScript 2015

Το ES6, επίσης γνωστό ως ECMAScript2015, εισήγαγε κλάσεις.

Μια κλάση είναι ένας τύπος συνάρτησης, αλλά αντί να χρησιμοποιήσουμε τη λέξη-κλειδί `function` για να την εισάγουμε, χρησιμοποιούμε τη λέξη-κλειδί `class` και οι ιδιότητες προσδιορίζονται μέσα σε μια μέθοδο `constructor()` .

Ορισμός κλάσης

Χρησιμοποιούμε τη λέξη-κλειδί `class` για να δημιουργήσουμε μια κλάση και προσθέτουμε πάντα τη `constructor()` μέθοδο.

Η μέθοδος κατασκευής καλείται κάθε φορά που αρχικοποιείται (initialized) το αντικείμενο κλάσης.

Παράδειγμα

Ένας απλός ορισμός κλάσης για μια κλάση με το όνομα "Car":

```
class Car {
  constructor(brand) {
    this.carname = brand;
  }
}
```

Τώρα μπορείτε να δημιουργήσετε αντικείμενα χρησιμοποιώντας την κατηγορία Car:

Παράδειγμα

Δημιουργήστε ένα αντικείμενο που ονομάζεται “mycar” με βάση την κλάση Αυτοκινήτου:

```
class Car {
  constructor(brand) {
    this.carname = brand;
  }
}
mycar = new Car("Ford");
```

Σημείωση: Η μέθοδος constructor καλείται αυτόματα όταν εισάγεται το αντικείμενο.

Μέθοδοι

Η μέθοδος constructor είναι ειδική, είναι εκεί όπου αρχικοποιείτε τις ιδιότητες, καλείται αυτόματα όταν ξεκινά μια κλάση και πρέπει να έχει το ακριβές όνομα “constructor”, στην πραγματικότητα, αν δεν έχετε μια μέθοδο κατασκευαστή, η JavaScript θα προσθέσει μια *αόρατη και κενή* μέθοδο κατασκευαστή.

Είστε επίσης ελεύθεροι να κάνετε τις δικές σας μεθόδους, η σύνταξη πρέπει να είναι οικεία:

Παράδειγμα

Δημιουργήστε μια μέθοδο που ονομάζεται “present”:

```
class Car {
  constructor(brand) {
    this.carname = brand;
  }
  present() {
    return "I have a " + this.carname;
  }
}

mycar = new Car("Ford");
document.getElementById("demo").innerHTML = mycar.present();
```

Όπως μπορείτε να δείτε στο παραπάνω παράδειγμα, καλείτε τη μέθοδο ανατρέχοντας στο όνομα της μεθόδου του αντικειμένου ακολουθούμενο από παρένθεση (οι παράμετροι θα μπουν μέσα στις παρενθέσεις).

Παράδειγμα

Αποστολή μιας παραμέτρου στη μέθοδο “present()”:

```
class Car {
  constructor(brand) {
    this.carname = brand;
  }
  present(x) {
    return x + ", I have a " + this.carname;
  }
}

mycar = new Car("Ford");
document.getElementById("demo").innerHTML =
```

Στατικές μέθοδοι

Οι στατικές μέθοδοι ορίζονται στην ίδια την κλάση και όχι στο πρωτότυπο.

Αυτό σημαίνει ότι δεν μπορείτε να καλέσετε μια στατική μέθοδο στο αντικείμενο (mycar), αλλά στην κλάση (Car):

Παράδειγμα

Δημιουργήστε μια στατική μέθοδο και καλέστε την στην κλάση:

```
class Car {
  constructor(brand) {
  this.carname = brand;
    }
  static hello() {
  return "Hello!!"; 7
    }
 }
 mycar=new Car("Ford");

//Call 'hello()' on the class Car:
document.getElementById("demo").innerHTML = Car.hello();
```

Εάν θέλετε να χρησιμοποιήσετε το αντικείμενο mycar μέσα στη στατική μέθοδο, μπορείτε να το στείλετε ως παράμετρο:

Παράδειγμα

Αποστολή “mycar” ως παράμετρος:

```
class Car {
  constructor(brand) {
    this.carname = brand;
  }
  static hello(x) {
    return "Hello " + x.carname;
  }
}

mycar = new Car("Ford");

document.getElementById("demo").innerHTML = Car.hello(mycar);
```

Κληρονομικότητα

Για να δημιουργήσετε κληρονομικότητα κλάσης, χρησιμοποιήστε την λέξη-κλειδί `extends` .

Μια κλάση που δημιουργήθηκε με κληρονομικότητα κλάσης κληρονομεί όλες τις μεθόδους από την άλλη κλάση:

Παράδειγμα

Δημιουργήστε μια κλάση που ονομάζεται “Model”, η οποία θα κληρονομήσει τις μεθόδους από την κατηγορία “Car”:

```
class Car {
  constructor(brand) {
    this.carname = brand;
```

```
  }
  present() {
    return 'I have a ' + this.carname;
  }
}

class Model extends Car {
  constructor(brand, mod) {
    super(brand);
    this.model = mod;
  }
  show() {
    return this.present() + ', it is a ' + this.model;
  }
}

mycar = new Model("Ford", "Mustang");
```

Η μέθοδος `super()` αναφέρεται στην κλάση γονέων.

Καλώντας τη μέθοδο `super()` στη μέθοδο του κατασκευαστή, καλούμε τη μέθοδο του κατασκευαστή γονέα και αποκτά πρόσβαση στις ιδιότητες και τις μεθόδους του γονέα.

Η κληρονομικότητα είναι χρήσιμη για την επαναχρησιμοποίηση κώδικα: επαναχρησιμοποίηση ιδιοτήτων και μεθόδων μιας υπάρχουσας κλάσης, όταν δημιουργείτε μια νέα κλάση.

Getters και Setters

Οι κλάσεις σας επιτρέπουν επίσης να χρησιμοποιήσετε getters και setters.

Μπορεί να είναι έξυπνο να χρησιμοποιείτε getters και setters για τις ιδιότητές σας, ειδικά εάν θέλετε να κάνετε κάτι ξεχωριστό με την τιμή πριν την επιστροφή τους ή πριν τις ορίσετε.

Για να προσθέσετε getters και setters στην κλάση, χρησιμοποιήστε τις λέξεις-κλειδιά `get` και `set` .

Παράδειγμα

Δημιουργήστε ένα getter και ένα setter για την ιδιότητα “carname”:

```
class Car {
  constructor(brand) {
  this._carname = brand;
    }
  get carname() {
```

```
    return this._carname;
  }
  set carname(x) {
    this._carname = x;
  }
}

mycar = new Car("Ford");
```

Σημείωση: ακόμα και αν η getter είναι μια μέθοδος, μην χρησιμοποιείτε παρενθέσεις όταν θέλετε να λάβετε την τιμή της ιδιότητας.

Το όνομα της μεθόδου getter/setter δεν μπορεί να είναι το ίδιο με το όνομα της ιδιότητας, στην περίπτωση αυτή `carname` .

Πολλοί προγραμματιστές χρησιμοποιούν ένα χαρακτήρα υπογράμμισης `_` πριν από το όνομα της ιδιότητας για να διαχωρίσουν τον getter/setter από την πραγματική ιδιότητα:

Παράδειγμα

Μπορείτε να χρησιμοποιήσετε τον χαρακτήρα υπογράμμισης για να διαχωρίσετε το getter/setter από την πραγματική ιδιότητα:

```
class Car {
  constructor(brand) {
    this._carname = brand;
  }
  get carname() {
    return this._carname;
  }
  set carname(x) {
    this._carname = x;
  }
}

mycar = new Car("Ford");

document.getElementById("demo").innerHTML = mycar.carname;
```

Για να χρησιμοποιήσετε ένα *setter* , χρησιμοποιήστε την ίδια σύνταξη όπως όταν ορίζετε μια τιμή ιδιότητας, χωρίς παρενθέσεις:

Παράδειγμα

Χρησιμοποιήστε έναν setter για να αλλάξετε την ονομασία “Volvo”:

```
class Car {
  constructor(brand) {
    this._carname = brand;
  }
  get carname() {
    return this._carname;
  }
  set carname(x) {
    this._carname = x;
  }
}

mycar = new Car("Ford");
mycar.carname = "Volvo";
document.getElementById("demo").innerHTML = mycar.carname;
```

Ανύψωση (hoisting)

Σε αντίθεση με τις συναρτήσεις και άλλες δηλώσεις της JavaScript, οι δηλώσεις κλάσης δεν ανυψώνονται.

Αυτό σημαίνει ότι πρέπει να δηλώσετε μια κλάση πριν τη χρησιμοποιήσετε:

Παράδειγμα

```
//You cannot use the class yet.
//mycar = new Car("Ford")
//This would raise an error.

class Car {
  constructor(brand) {
    this.carname = brand;
  }
}

//Now you can use the class:
mycar = new Car("Ford")
```

Σημείωση: Για άλλες δηλώσεις, όπως συναρτήσεις, ΔΕΝ θα λάβετε σφάλμα όταν προσπαθείτε να τις χρησιμοποιήσετε πριν να δηλωθούν, επειδή η προεπιλεγμένη συμπεριφορά των δηλώσεων JavaScript ανυψώνεται (μετακινώντας τη δήλωση στην κορυφή).

“use strict”;

Η σύνταξη στις κλάσεις πρέπει να γράφεται σε “αυστηρή λειτουργία”.

Θα λάβετε ένα σφάλμα αν δεν ακολουθείτε τους κανόνες “αυστηρής λειτουργίας”.

Παράδειγμα

Στην “αυστηρή λειτουργία” θα λάβετε ένα σφάλμα αν χρησιμοποιήσετε μια μεταβλητή χωρίς να δηλώσετε:

```
class Car {
  constructor(brand) {
    i = 0;
    this.carname = brand;
  }
}
var mycar = new Car("Ford");
```