Util module

The node.js "util" module provides some functions to print formatted strings as well as some 'utility' functions that are helpful for debugging purposes.

Use require('util') to access these functions.

Functions:

1) util.format(format, [...])

The util.format() is used to create formatted string from one or more arguments. The first argument is a string that contains zero or more placeholders. A placeholder is a character sequence in the format string and is replaced by a different value in the returned string.

List of placeholder:

% - Returns a single percent sign.

d - Treated as Number (both integer and float).

s - Treated as string and display as a string.

j - Represent JSON data.

Example:

```
var util = require('util');

var my_name = 'Sunita',

my_class = 5,

my_roll_no = 11,

my_fav_subject= { subj1: 'English', subj2: 'Math.'};

var format1 = util.format('My name is %s ',my_name);

var format2 = util.format('I read in class %d,',my_class);

var format3 = util.format('My roll no. is %d,',my_roll_no);

var format4 = util.format('My favorite subjects are %j',my_fav_subject);

console.log(format1);

console.log(format2);

console.log(format3);

console.log(format4);
```

Output:

```
E:\nodejs>node test.js

My name is Sunita

I read in class 5,

My roll no. is 11,

My favorite subjects are {"subj1":"English","subj2":"Math."}
```

2) util.debug(string)

The function is used to block the process and output string immediately to stderr.

Example:

```
var util = require('util');

var testString = "Test Test";

util.debug(testString); // "Test Test";


var test = {};

util.debug(test); // "[object Object]";

util.debug(JSON.stringify(test)); // "{}"
```

Output:

```
E:\nodejs>node test.js

DEBUG: Test Test

DEBUG: [object Object]

DEBUG: {}
```

3) util.error([...])

The function accepts multiple arguments and writes them out to stderr.

Example:

```
var util = require("util");

util.error("Error-1","Error-2","Error-3");
```

Output:

```
E:\nodejs>node test.js

Error-1

Error-2

Error-3
```

4) util.puts([...])

The function accepts multiple arguments and writes them out to stderr with newlines after each argument.

Example:

```
var util = require("util");

util.puts("A", "B","C");
```

Output:

E:\nodejs>node test.js

A

B

C

5) util.print([...])

The function accepts multiple arguments, converts each one to a string and then writes them out to stdout without adding a new line after each argument.

Example:

```
var util = require("util");

util.print(1, 2, '3');
```

Output:

E:\nodejs>node test.js

123

6) util.log(string)

The function is used to write the string out to stdout, with timestamp.

Example:

```
var util = require('util');

util.log('Timestamped message.');
```

Output:

E:\nodejs>node test.js

24 Oct 14:23:16 - Timestamped message.

7) util.inspect(object, [options])

The function returns a string representation of object, which is useful for debugging.

Optional options:

showHidden - if true then the object's non-enumerable properties will be shown too. Defaults to false.

depth - tells inspect how many times to recurse while formatting the object. This is useful for inspecting large complicated objects. Defaults to 2. To make it recurse indefinitely pass null.

colors - if true, then the output will be styled with ANSI color codes. Defaults to false. Colors are customizable, see below.

customInspect - if false, then custom inspect() functions defined on the objects being inspected won't be called. Defaults to true.

The following example lists the Node's built-in objects.

```
var util = require('util')

console.log(util.inspect(console));
```

Output:

```
E:\nodejs>node test.js

{ log: [Function],
  info: [Function],
  warn: [Function],
  error: [Function],
  dir: [Function],
  time: [Function],
  timeEnd: [Function],
  trace: [Function],
  assert: [Function],
  Console: [Function: Console] }
```

Here is an example of inspecting all properties of the util object :

```
var util = require('util');

console.log(util.inspect(util, { showHidden: true, depth: null }));
```

Customizing util.inspect colors :

The optional argument colorize is a boolean that adds ANSI escape codes to the string output. When logged to a terminal window, it should be pretty printed with colors.

```
var util = require('util');

console.log(util.inspect({x:100, y:"y"}, false,2,true));
```

Output:

```
util.inspect
```

8) util.isArray(object)

The function is used to check whether an 'object' is an array or not. Returns true if the given 'object' is an Array, false otherwise.

Example:

```
var util = require('util');

console.log(util.isArray([]));

console.log(util.isArray(new Array));

console.log(util.isArray({}))
```

Output:

```
E:\nodejs>node test.js

true

true

false
```

9) util.isRegExp(object)

The function is used to check whether an 'object' is RegExp or not. Returns true if the given 'object' is an RegExp, false otherwise.

Example:

```
var util = require('util');

console.log(util.isRegExp(/some regexp/));

onsole.log(util.isRegExp(new RegExp('New regexp')));

console.log(util.isRegExp({}))
```

Output:

```
E:\nodejs>node test.js

true

true

false
```

10) util.isDate(object)

The function is used to check whether an 'object' is Date or not. Returns true if the given 'object' is an Date, false otherwise.

Example:

```
var util = require('util');

console.log(util.isDate(new Date()));

console.log(util.isDate(Date()));

console.log(util.isDate({}))
```

Output:

```
E:\nodejs>node test.js

true

false

false
```

11) util.isError(object)

The function is used to check whether an 'object' is Error or not. Returns true if the given 'object' is an Error, false otherwise.

Example:

```
var util = require('util');

console.log(util.isError(new Error()));

console.log(util.isError(new TypeError()));

console.log(util.isError({ name: 'Error', message: 'an error occurred' }));
```

Output:

```
E:\nodejs>node test.js

true

true

false
```

12) util.inherits(constructor, superConstructor)

The function is used to inherit the prototype methods from one constructor into another. The prototype of constructor will be set to a new object created from superConstructor. As an additional convenience, superConstructor will be accessible through the constructor.super_ property.

Example:

```
var util = require("util");

var events = require("events");

function MyStream() {

   events.EventEmitter.call(this);

}

util.inherits(MyStream, events.EventEmitter);

MyStream.prototype.write = function(data) {

   this.emit("data", data);

}
```

```
var stream = new MyStream();
console.log(stream instanceof events.EventEmitter);
console.log(MyStream.super_ === events.EventEmitter);
stream.on("data", function(data) {
   console.log('Received data: "' + data + '"');
})
stream.write("It works!");
```

Output:

```
E:\nodejs>node test.js
true
true
Received data: "It works!"
```