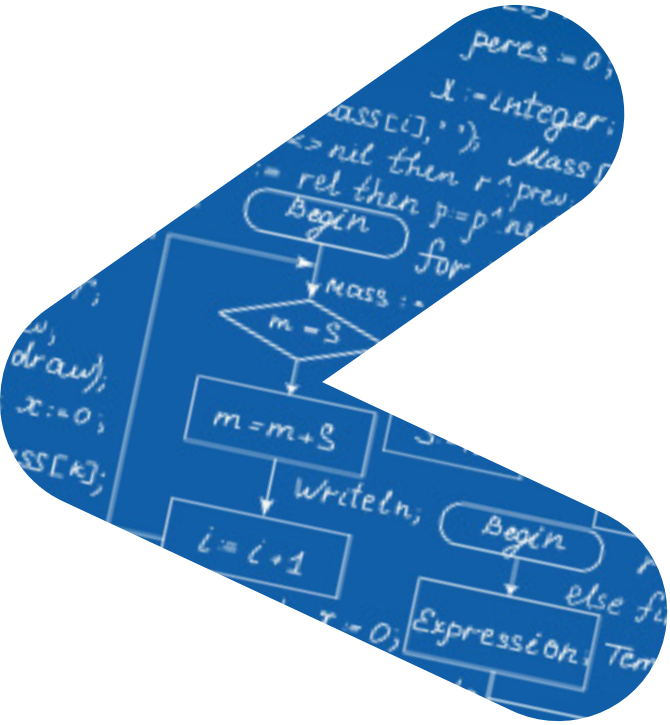
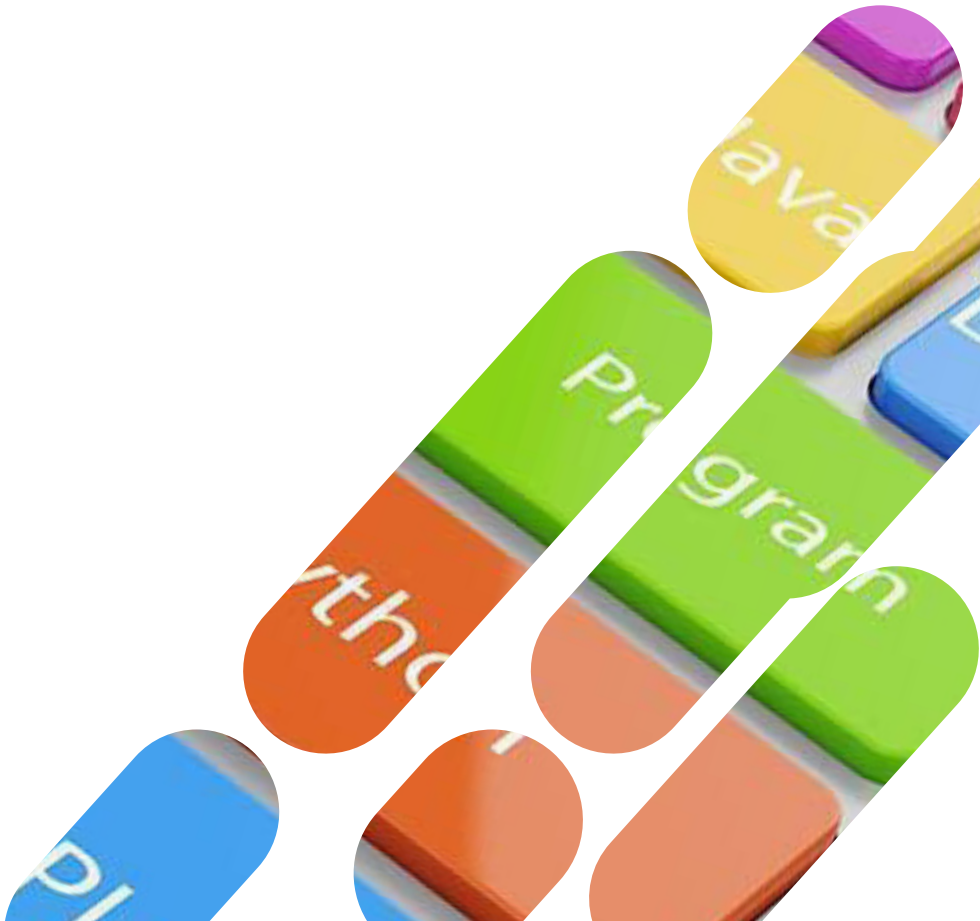


SMK Telkom Makassar

ALGORITMA
PEMROGRAMAN



PEMROGRAMAN TERSTRUKTUR

PENGKONDISIAN

Rekaya Perangkat Lunak SMK Telkom Makas*sar*

Tujuan Pembelajaran

1. Peserta didik dapat Memahami struktur pengkondisian
2. Perserta didik dapat Membuat Program Sederhana Dengan Pengkondisian

Komputer beroperasi dengan cara yang sangat logis. Beberapa jawaban favorit komputer adalah "**ya**" dan "**tidak**". Dalam istilah komputer, ini setara dengan "benar" dan "salah". Kemampuan untuk membedakan antara benar dan salah adalah kekuatan yang luar biasa. Ini berarti komputer dapat memutuskan kode apa yang harus dijalankan tergantung pada situasinya. Ini juga berarti bahwa satu program dapat melakukan banyak hal dan membuat keputusan sendiri! Keputusan-keputusan tersebut dikontrol dengan menggunakan kondisional.

Pemeriksaan dan Validasi Data

Sebelum memanfaatkan kekuatan kondisional, strategi yang digunakan untuk membuat pengambilan keputusan yang tepat perlu dibahas. Nilai mentah benar dan salah tidak umum digunakan dalam kondisional. Sebaliknya, mereka hampir selalu dihitung melalui operasi lain.

Operator Pembanding

Operasi perbandingan paling umum yang akan dilakukan adalah memeriksa apakah dua nilai identik atau tidak identik. Operator kesetaraan, dilambangkan sebagai tanda sama dengan, memeriksa apakah dua nilai identik.

```
10 == 10; // true
'pizza' == 'pizza'; // true

'pizza' == 'Pizza'; // false
false == false; // true
```

Siap untuk kejutan yang tidak menyenangkan? Operator kesetaraan akan menggunakan pemaksaan tipe jika memungkinkan

```
'10' == 10; // true
```

Disarankan untuk tidak menggunakan operator persamaan. Operator yang berbeda melakukan pekerjaan yang lebih baik untuk tujuan yang sama.

OPERATOR KESETARAAN YANG KETAT

Operator kesetaraan ketat adalah saudara kandung operator kesetaraan yang lebih logis, dilambangkan sebagai tanda sama dengan tiga. Operator ini memeriksa kesetaraan dan tidak pernah menggunakan pemaksaan tipe.

```
10 === 10; // true
'pizza' === 'pizza'; // true
'pizza' === 'Pizza'; // false
```

```
false === false; // true
// no type coercion performed
‘10’ === 10; // false
```

OPERATOR KETIDAKSETARAAN
Operator pertidaksamaan, dilambangkan sebagai tanda seru dan tanda sama dengan, memeriksa apakah dua nilai tidak identik.

```
10 != 10; // false
‘pizza’ != ‘pizza’; // false
‘pizza’ != ‘Pizza’; // true
false != false; // false
```

Sama seperti operator persamaan, operator pertidaksamaan akan menggunakan pemaksaan tipe.

```
‘10’ != 10; // false
```

OPERATOR KETIDAKSETARAAN YANG KETAT
Operator pertidaksamaan ketat adalah versi yang lebih baik dari operator pertidaksamaan. Sama seperti persamaan ketat, pertidaksamaan ketat tidak melakukan pemaksaan jenis apa pun. Dibandingkan dengan pertidaksamaan biasa, operator ini lebih aman untuk kode JavaScript. Sintaksnya dilambangkan dengan tanda seru dan tanda sama dengan dua. Silakan gunakan ini sebagai pengganti operator pertidaksamaan biasa.

```
10 !== 10; // false
‘pizza’ !== ‘pizza’; // false
‘pizza’ !== ‘Pizza’; // true
false !== false; // false
// no type coercion performed
‘10’ !== 10; // true
```

Operator Relasional

Dengan cara yang sama, dua nilai dapat diperiksa kesetaraan atau ketidaksetaraannya, nilai dapat diperiksa apakah lebih besar dari atau lebih kecil dari.

KURANG DARI
Dua operator perbandingan yang tersedia adalah operator kurang dari dan operator kurang dari atau sama dengan. Keduanya berfungsi persis seperti namanya. Angka dibandingkan secara numerik, dan string dibandingkan menurut abjad. Penting untuk dicatat bahwa kapitalisasi penting,

```
10 < 8; // false
```

```
'a' < 'b'; // true
'A' > 'a'; // false
10 <= 8; // false
```

LEBIH BESAR DARI OPERATOR
Operator lebih besar dari dan lebih besar dari atau sama dengan operator memiliki prinsip yang sama dengan operator kurang dari, dan kapitalisasi masih penting.

```
10 > 8; // true
'a' > 'b'; // false
'a' >= 'b'; // true
```

Operator Logika

Selain operator kesetaraan dasar, JavaScript menyediakan operator khusus yang membantu membuat nilai boolean dalam situasi yang kompleks. Banyak operator logika dan operator kesetaraan yang dapat dipadukan dan dicocokkan.

OR
Operator OR, dilambangkan sebagai pipa ganda, memeriksa apakah salah satu operasi tersebut benar.

```
true || false; // true
false || false; // false
false || true || false; // true
```

AND
Operator AND, dilambangkan sebagai tanda pagar ganda, memeriksa untuk melihat apakah kedua operasi menghasilkan nilai benar.

```
true && true; // true
true && false; // false
false && false; // false
false && true && false; // false
```

NOT
Dalam pemrograman, ada juga konsep operator NOT, yang dilambangkan sebagai tanda seru di depan sebuah nilai. Operator ini biasanya disebut sebagai operator bang. Operator ini membalik nilai boolean dari nilai yang dioperasikan.

```
!true; // false
!false; // true
```

Padu Padankan Operator Logika dan Persamaan

Kata kunci true dan false jarang digunakan dalam kode. Sekarang setelah Anda memiliki operator pembanding kesetaraan dan operator logika dalam tool kit Anda, Anda akan melihat kondisi yang berkisar dari yang sederhana hingga yang kompleks dalam kode.

```
// 1 is greater than 10 OR 1 is equal to 0
1 > 10 || 1 === 0; // false
// 1 is less than 10 OR 1 is equal to 0
1 < 10 || 1 === 0; // true
```

Validasi Data

Checking if a value is undefined or null is easy.

```
var mySocialLife = null;
mySocialLife === null; // true
```

Operator persamaan menanganinya, karena tipe data tersebut memiliki satu kemungkinan nilai. Tetapi bagaimana dengan angka, string, boolean, dan array? Bagaimana jika Anda tidak tertarik dengan nilai dari sebuah angka? Mungkin Anda hanya ingin mengetahui apakah itu angka atau bukan. Untungnya, ada operator untuk situasi ini.

TYPEOF OPERATOR
Operator typeof mengeluarkan tipe data variabel, dalam bentuk string. Hal ini sangat membantu ketika men-debug kode. Tipe data yang salah adalah sumber kesalahan yang umum terjadi!

```
typeof 10; // ‘number’
typeof ‘’; // ‘string’
typeof undefined; // ‘undefined’
typeof true; // ‘boolean’
```

Hal yang sederhana. Tapi izinkan saya melemparkan sebuah bola lengkung kepada Anda.

```
typeof null; // ‘object’—what????
```

Hah? Mengapa typeof null tidak menghasilkan string null? Masih bisa diperdebatkan apakah ini bug atau bukan. Tetapi pada akhirnya, ini sangat tidak intuitif. Sebagai gantinya, periksa null dengan menggunakan operator persamaan.

```
null === null // true
```

CONTOH DARI OPERATOR
Ketika memeriksa apakah sebuah item adalah sebuah larik, operator typeof sangat menipu.

```
typeof []; // ‘object’
```

Alasan yang mendasari mengapa keluarannya adalah “objek” secara teknis masuk akal, tetapi tidak membantu dan tidak intuitif.

Validasi larik dapat dilakukan dengan benar menggunakan operator instanceof.

```
[] instanceof Array; // true
```

Perhatikan bahwa Array bukanlah sebuah string dalam situasi ini. Hal ini berbeda dengan typeof, yang menghasilkan sebuah string.

Stetmen Kondisi

Setelah mempelajari semua cara untuk memvalidasi data dan membandingkan nilai, Anda siap menggunakannya untuk menyelesaikan tantangan yang berarti. Pernyataan bersyarat yang paling umum adalah pernyataan if . . . else if . . . else.

Pernyataan Percabangan Bersyarat

Istilah "pernyataan percabangan bersyarat" adalah cara yang bagus untuk menggambarkan pernyataan if . . . else if . . . else. Hal ini memungkinkan seorang programmer untuk melakukan tindakan berdasarkan kondisi tertentu.

IF STATEMENT
Pernyataan if menjalankan kode jika kondisinya bernilai benar. Kata kuncinya adalah if, dengan tanda kurung di sekitar kondisi dan tanda kurung kurawal yang membungkus operasi. Sejumlah operasi dapat ditempatkan di dalam pernyataan if. Perhatikan bahwa tidak ada titik koma.

```
if (condition) {
// do something here
}
```

Mari kita memodifikasi string secara kondisional, berdasarkan nilainya.

```
var catName = 'Max';
if (catName === 'Max') {
   catName += ' is an awesome cat';
}
catName; // 'Max is an awesome cat'
```

ELSE STATEMENT
Bagaimana jika Anda ingin mengekspresikan penghinaan Anda terhadap kucing yang tidak bernama Max? Di situlah pernyataan else berperan. Setelah pernyataan if, pernyataan else dapat dirantai ke pernyataan tersebut. Jika kondisi if gagal, maka kode pernyataan else akan dijalankan.

```
var catName = ‘Layna’;
if (catName === ‘Max’){
   catName += ‘ is an awesome
   cat’;
 } else {
   catName += ‘ is NOT an awesome cat’;
 }
catName; // ‘Layna is NOT an awesome cat’
```

ELSE IF STATEMENT

Bagaimana jika Anda juga memutuskan bahwa Charlie adalah nama yang dapat diterima untuk seekor anjing? Pernyataan else if dapat ditambahkan ke rantai kondisional. Pernyataan ini mengikuti sintaks yang sama dengan pernyataan if.

```
var catName = ‘Charlie’;
if (catName === ‘Max’) {
   catName += ‘ is an awesome
   cat’;
 }else if (catName ===
 ‘Charlie’) {
   catName += ‘ is an okay cat’;
 }else{
   catName += ‘ is NOT an awesome
   cat’;
}
catName; // ‘Charlie is an okay cat’
```

Conditionals in Action

```
var prompt = require(‘readline- sync’);
var jenis = prompt.question(‘apakah jenis kucing
anda? ‘);
var kucing = {
lokal: ‘Suzy’,
anggora: ‘pus’,
persia: ‘boy’
};
 console.log(‘kucingmu bernama
 ‘ + jenis[kucing] + ‘? Bagaimana menarik!!’);
```

Apabila jenis tidak ada dalam objek, output tidak terdefinisi. Itu bukan representasi yang baik dari apa yang terjadi, karena masalah sebenarnya adalah bahwa objek kucing tidak mencakup semua jenis kucing yang ada di planet ini. Kode harus mengkomunikasikan hal ini. Dengan menggunakan kondisional, mari kita selesaikan masalah sebelumnya! Sebelum mengimplementasikannya, mari kita tulis beberapa kode semu.

KONDISIONAL MEMILIKI BEBERAPA PENDEKATAN

Untuk algoritma yang melibatkan kondisional, ada banyak cara untuk mengatakan hal yang sama. Berikut adalah dua cara berbeda untuk mengatakan hal yang sama:

```
“Jika kita menemukan jenis kucing di dalam objek
 maka cetak: nama kucing kita.
 Jika tidak, cetak: kucing tidak ditemukan
 Jika kita tidak menemukan jenis, maka
 cetak: kucing tidak ditemukan. Lain,
 cetak: nama kucing kita “
```

Jika dipikir-pikir, ini adalah dua sisi dari koin yang sama. Mereka mengatakan hal yang sama persis, kecuali dalam urutan yang berlawanan. Mari kita ungkapkan dua pernyataan yang mungkin Anda buat dengan menggunakan sintaksis JavaScript.

```
if (kita menemukan sebuah nilai) {
   cetak nama kucing kita
 } else {
   cetak kucing tidak ditemukan
 }
 if (kita tidak menemukan sebuah nilai) {
   cetak kucing tidak ditemukan
 } else {
   cetak nama kucing kita
}
```

Menerjemahkan logika ini mengharuskan Anda untuk berpikir seperti komputer. Anda membutuhkan JavaScript yang setara dengan “mencari nilai”. Mengakses sebuah kunci dalam objek yang tidak ada, Anda akan mendapatkan nilai yang tidak terdefinisi. Jika nilainya tidak terdefinisi, itu menyiratkan bahwa kuncinya tidak ada. Ada dua cara yang berbeda untuk menulis ini.

typeof value === ‘undefined’; value === undefined;