CEP - UP - UTFPR - PUCPr – 27/05/2024 - 10:43:53.8
Prof Dr P. Kantek (pkantek@gmail.com)
O JS como linguagem desktop VIVXh61a, V: 1.05
76137 PEDRO HENRIQUE PIEKARSKI DE(76137);
LUIZ MIGUEL OLINEK (76106);
ERIC CORDEIRO (76025);
24FRO203 - 1 entregar ate 11/jun _____ / / _____

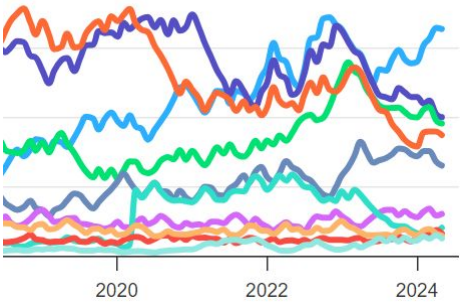
JS como linguagem de desktop

O Javascript tem como vocação principal dar vitalidade a páginas HTML, permitindo que elas ganhem um comportamento dinâmico e sobretudo possam interagir como o usuário da página sendo exibida. Entretanto, graças à tecnologia que floresceu no entorno do javascript, ele acabou se tornando um ambiente de desenvolvimento/produção completo. Esta folha aborda este conceito, que é o de disponibilizar (mais) uma bancada completa para o programador desenvolver sua arte. Relembre que o compilador/interpretador Javascript está dentro do browser e só tem sentido falar em JS ao usar uma sessão desse programa.

Aliás, um parênteses oportuno: o que é melhor:

- Privilegiar uma única linguagem se tornando **o** especialista nela e conhecendo **todos** ou quase todos os meandros e idiossincrasias dela OU
- Conhecer e usar um leque (crescente) de linguagens, plataformas, ambientes diversos, alternando usos, e procurando de cada um o que ele faz bem.

Não tenho resposta para isto, mas a dúvida é propícia para alguns raciocínios. O primeiro enfoque te permite ser um especialista, o que em muitos casos é fortemente buscado pelo mercado de empregos (por exemplo). Além disso, há funcionalidades que só podem ser boladas e desenvolvidas por este personagem. Mas, um problema aqui é a obsolescência meio programada desta expertise. Se você olhar a história recente da informática, verá que inúmeros ambientes já foram a crista da onda e em poucos anos deixaram de sê-lo. Lá atrás, este papel foi assumido pelo Fortran, e depois pelo Cobol e hoje tais linguagem no mínimo, cheiram a naftalina. Mais recentemente o Java pode ser colocado nesse balaio. O queridinho da hora é o Python, mas provavelmente o é até que surja o novo objeto de desejo de todos.



Neste gráfico (TIOBE de maio de 2024) considerado o melhor termômetro de uso de linguagens no mundo, o azul claro é Python, o azul escuro é C, o verde claro é C++ e o laranja é Java. A propósito, o cor de rosa é Javascript.

Já a segunda abordagem nos permite uma maior segurança, já que não coloca todos os ovos na mesma cesta. Mas, não existe almoço grátis, ao transitar por diversos ambientes, há uma dificuldade adicional: a confusão entre eles, e até uma certa complexidade em traduzir um único algoritmo em diversas encarnações. Além do que, é quase impossível ser um completo expert em plataformas diferentes. Mas, de qualquer forma, eu prefiro sempre um poliglota a um monoglota, por mais competente que seja.

Isto posto, vamos olhar o Javascript como um faz-tudo.

Declaração de Variáveis:

var Declara uma variável com escopo de função.

let Declara uma variável com escopo de bloco, ou seja tudo o que está entre { e }.

const Declara uma variável constante (?) com escopo de bloco.

Tipos de Dados:

number Números inteiros e decimais.

string Textos entre aspas simples ou duplas.

boolean Valores True ou False.

array Coleções ordenadas de dados.

object Coleções não ordenadas de dados chave-valor.

Operadores:

Aritméticos (`+`, `-`, `*`, `/`, `%`): Realizam operações matemáticas básicas.

Comparação (`==`, `!=`, `<`, `>`, `<=`, `>=`): Comparação de valores.

Lógicos (`&&`, `||`, `!`): Operações lógicas AND, OR e NOT.

Estruturas de Controle:

if/else Executa um bloco de código se uma condição for verdadeira, senão outro bloco.

switch/case Executa um bloco de código específico com base em uma comparação.

while Repete um bloco de código enquanto uma condição for verdadeira.

for Repete um bloco de código um número específico de vezes.

Funções:

function Cria uma função reutilizável para executar um bloco de código.

return Retorna um valor da função.

Repare que até aqui, não se falou uma única vez de HTML, coerente com o objetivo desta folha.

Ainda falta a entrada e saída, que aqui é meio complicada, afinal não nos esqueçamos que tem um browser no meio da comunicação. Para falar, (output) o programa pode lançar mão de `alert(msg)` que interrompe o programa e lança uma caixa de diálogo (em forma modal). Há também o `var=prompt(msg)` que faz o mesmo mas ainda recupera o que o usuário digitou colocando-o em `var`. O alert interrompe o fluxo do programa que só é retomado após o usuário apertar o botão `OK`. Já o prompt, devolve o que o usuário digitar (se ele apertar `OK`) ou retorna null se ele apertar `Cancelar`).

Veja um exemplo de tudo o que se disse acima. Estude o programa a seguir

Exemplo: Jogo da adivinhação

Este jogo simples desafia o usuário a adivinhar um número secreto gerado aleatoriamente pelo computador. O jogo fornece feedback ao usuário, indicando se o palpite está muito alto, muito baixo ou correto.

```
// Gera um número aleatório entre 1 e 100
let numSecr=Math.floor(Math.random()*100)+1;
// Define o número máximo de tentativas
let tentMax = 10;
// Mantém o controle do número de tentativas
let tentRest = tentMax;
// Loop principal do jogo
while (tentRest > 0) {
  // Obtém o palpite do usuário
  let palpite = prompt(`Digite um número 1..100
  (tentativas restantes: ${tentRest})`);
  // Verifica se o palpite é válido
  if (isNaN(palpite)||palpite<1||palpite>100) {
    alert("Ruim: Digite um número entre 1 e 100.");
    continue;
  }
  // Converte o palpite para um número
  palpite = parseInt(palpite);
  // Verifica se o palpite está correto
  if (palpite === numSecr) {
    alert(`Parabéns! Você adivinhou  em
    ${tentMax - tentRest + 1} tentativas.`);
    break;
  } else if (palpite < numSecr) {
    alert("O número é maior que seu palpite.");
  } else {
    alert("O número é menor que seu palpite.");
  }
  // Decrementa o número de tentativas restantes
  tentRest--;
}
// Informa a mensagem final caso não tenha
adivinhado o número
if (tentRest === 0) {
  alert(`Você não conseguiu adivinhar em
  ${tentMax} tentativas. Ele era ${numSecr}.`);
}
```

Há aqui a construção `{...}` que é conhecida como interpolação de string, é uma ferramenta para formatar strings de forma dinâmica e expressiva. Ela permite incorporar expressões JavaScript dentro de strings literais, resultando em strings personalizadas e dinâmicas.

A construção `${...}` é composta por um par de chaves ({}) que contêm uma expressão JavaScript. A expressão é avaliada e seu valor é inserido na string literal no local onde a construção aparece. Isso permite que você combine strings estáticas com dados dinâmicos gerados a partir de variáveis, funções ou outras operações JavaScript.

o operador === representa a comparação estrita entre dois valores. Ele verifica se os valores dos operandos (elementos que o operador compara) são iguais e do mesmo tipo.

JavaScript possui outro operador de comparação, ==, que realiza uma comparação solta. Isso significa que ele pode converter implicitamente os valores dos operandos para o mesmo tipo antes da comparação. Por exemplo, 1 == ”1” retorna true porque 1 (número) é convertido para a string ”1” para realizar a comparação.

`isNaN()` é uma função global usada para determinar se um valor é Not a Number (Não é um Número). Ela verifica se o valor passado como argumento representa um NaN (Not a Number) especial.

`parseInt()` é uma função usada para analisar uma string e convertê-la em um número inteiro (inteiro). Ela analisa o início da string fornecida e extrai o número inteiro a partir dela, considerando uma base específica (sistema de numeração) opcional. Neste caso seria `parseInt(string, base)`

`continue` e `break` funcionam exatamente como em outras linguagens como C++ ou Python: o primeiro força a reanálise da condição do laço, enquanto o segundo força uma saída incondicional do mesmo.

Finalmente, faltou dizer que numa sessão do browser, se você apertar F12, vai abrir uma janela na qual apertando a aba CONSOLE você terá uma janela para emitir comandos JS e obter eventuais respostas.

☞ Para você fazer

Você deve digitar/copiar este programa, salvando-o no mesmo diretório de trabalho onde vai criar depois o HTML que o chamará.

Use um nome qualquer para este programa, digamos `XX.JS`.

A seguir, escreva um HTML simples como em

```
<html>
<head> <meta charset="UTF-8">
<script src="xx.js"> </script> </head>
</html>
```

Execute o programa e o HTML clicando neste último. Veja o desempenho do programa.

Apenas para aprender mais uma, exclua a cláusula `<meta charset="UTF-8">` e reexecute o programa. O que aconteceu ?

Mostre os arquivos gerados no computador ao professor, OU alternativamente imprima os arquivos pedidos grampeie-os nesta folha e devolva tudo ao professor.

Avaliação:

203-76137 - ga/ a

CEP - UP - UTFPR - PUCPr – 27/05/2024 - 10:43:53.8
Prof Dr P. Kantek (pkantek@gmail.com)
O JS como linguagem desktop VIVXh61a, V: 1.05
76049 FELIPE JESMIEL LEITE (76049);
VICTOR BRONDANI DOS SANTOS (76168);
GUILHERME VITOR DA SILVA (76063);
24FRO203 - 2 entregar ate 11/jun _____ / / _____

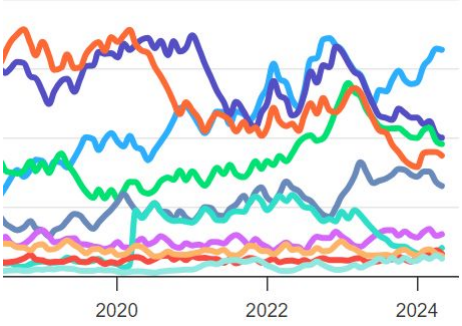
JS como linguagem de desktop

O Javascript tem como vocação principal dar vitalidade a páginas HTML, permitindo que elas ganhem um comportamento dinâmico e sobretudo possam interagir como o usuário da página sendo exibida. Entretanto, graças à tecnologia que floresceu no entorno do javascript, ele acabou se tornando um ambiente de desenvolvimento/produção completo. Esta folha aborda este conceito, que é o de disponibilizar (mais) uma bancada completa para o programador desenvolver sua arte. Relembre que o compilador/interpretador Javascript está dentro do browser e só tem sentido falar em JS ao usar uma sessão desse programa.

Aliás, um parênteses oportuno: o que é melhor:

- Privilegiar uma única linguagem se tornando **o** especialista nela e conhecendo **todos** ou quase todos os meandros e idiossincrasias dela OU
- Conhecer e usar um leque (crescente) de linguagens, plataformas, ambientes diversos, alternando usos, e procurando de cada um o que ele faz bem.

Não tenho resposta para isto, mas a dúvida é propícia para alguns raciocínios. O primeiro enfoque te permite ser um especialista, o que em muitos casos é fortemente buscado pelo mercado de empregos (por exemplo). Além disso, há funcionalidades que só podem ser boladas e desenvolvidas por este personagem. Mas, um problema aqui é a obsolescência meio programada desta expertise. Se você olhar a história recente da informática, verá que inúmeros ambientes já foram a crista da onda e em poucos anos deixaram de sê-lo. Lá atrás, este papel foi assumido pelo Fortran, e depois pelo Cobol e hoje tais linguagem no mínimo, cheiram a naftalina. Mais recentemente o Java pode ser colocado nesse balaio. O queridinho da hora é o Python, mas provavelmente o é até que surja o novo objeto de desejo de todos.



Neste gráfico (TIOBE de maio de 2024) considerado o melhor termômetro de uso de linguagens no mundo, o azul claro é Python, o azul escuro é C, o verde claro é C++ e o laranja é Java. A propósito, o cor de rosa é Javascript.

Já a segunda abordagem nos permite uma maior segurança, já que não coloca todos os ovos na mesma cesta. Mas, não existe almoço grátis, ao transitar por diversos ambientes, há uma dificuldade adicional: a confusão entre eles, e até uma certa complexidade em traduzir um único algoritmo em diversas encarnações. Além do que, é quase impossível ser um completo expert em plataformas diferentes. Mas, de qualquer forma, eu prefiro sempre um poliglota a um monoglota, por mais competente que seja.

Isto posto, vamos olhar o Javascript como um faz-tudo.

Declaração de Variáveis:

var Declara uma variável com escopo de função.

let Declara uma variável com escopo de bloco, ou seja tudo o que está entre { e }.

const Declara uma variável constante (?) com escopo de bloco.

Tipos de Dados:

number Números inteiros e decimais.

string Textos entre aspas simples ou duplas.

boolean Valores True ou False.

array Coleções ordenadas de dados.

object Coleções não ordenadas de dados chave-valor.

Operadores:

Aritméticos (`+`, `-`, `*`, `/`, `%`): Realizam operações matemáticas básicas.

Comparação (`==`, `!=`, `<`, `>`, `<=`, `>=`): Comparação de valores.

Lógicos (`&&`, `||`, `!`): Operações lógicas AND, OR e NOT.

Estruturas de Controle:

if/else Executa um bloco de código se uma condição for verdadeira, senão outro bloco.

switch/case Executa um bloco de código específico com base em uma comparação.

while Repete um bloco de código enquanto uma condição for verdadeira.

for Repete um bloco de código um número específico de vezes.

Funções:

function Cria uma função reutilizável para executar um bloco de código.

return Retorna um valor da função.

Repare que até aqui, não se falou uma única vez de HTML, coerente com o objetivo desta folha.

Ainda falta a entrada e saída, que aqui é meio complicada, afinal não nos esqueçamos que tem um browser no meio da comunicação. Para falar, (output) o programa pode lançar mão de `alert(msg)` que interrompe o programa e lança uma caixa de diálogo (em forma modal). Há também o `var=prompt(msg)` que faz o mesmo mas ainda recupera o que o usuário digitou colocando-o em `var`. O alert interrompe o fluxo do programa que só é retomado após o usuário apertar o botão `OK`. Já o prompt, devolve o que o usuário digitar (se ele apertar `OK`) ou retorna null se ele apertar `Cancelar`).

Veja um exemplo de tudo o que se disse acima. Estude o programa a seguir

Exemplo: Jogo da adivinhação

Este jogo simples desafia o usuário a adivinhar um número secreto gerado aleatoriamente pelo computador. O jogo fornece feedback ao usuário, indicando se o palpite está muito alto, muito baixo ou correto.

```
// Gera um número aleatório entre 1 e 100
let numSecr=Math.floor(Math.random()*100)+1;
// Define o número máximo de tentativas
let tentMax = 10;
// Mantém o controle do número de tentativas
let tentRest = tentMax;
// Loop principal do jogo
while (tentRest > 0) {
  // Obtém o palpite do usuário
  let palpite = prompt(`Digite um número 1..100
  (tentativas restantes: ${tentRest})`);
  // Verifica se o palpite é válido
  if (isNaN(palpite)||palpite<1||palpite>100) {
    alert("Ruim: Digite um número entre 1 e 100.");
    continue;
  }
  // Converte o palpite para um número
  palpite = parseInt(palpite);
  // Verifica se o palpite está correto
  if (palpite === numSecr) {
    alert(`Parabéns! Você adivinhou  em
    ${tentMax - tentRest + 1} tentativas.`);
    break;
  } else if (palpite < numSecr) {
    alert("O número é maior que seu palpite.");
  } else {
    alert("O número é menor que seu palpite.");
  }
  // Decrementa o número de tentativas restantes
  tentRest--;
}
// Informa a mensagem final caso não tenha
adivinhado o número
if (tentRest === 0) {
  alert(`Você não conseguiu adivinhar em
  ${tentMax} tentativas. Ele era ${numSecr}.`);
}
```

Há aqui a construção `{...}` que é conhecida como interpolação de string, é uma ferramenta para formatar strings de forma dinâmica e expressiva. Ela permite incorporar expressões JavaScript dentro de strings literais, resultando em strings personalizadas e dinâmicas.

A construção `${...}` é composta por um par de chaves ({}) que contêm uma expressão JavaScript. A expressão é avaliada e seu valor é inserido na string literal no local onde a construção aparece. Isso permite que você combine strings estáticas com dados dinâmicos gerados a partir de variáveis, funções ou outras operações JavaScript.

o operador `===` representa a comparação estrita entre dois valores. Ele verifica se os valores dos operandos (elementos que o operador compara) são iguais e do mesmo tipo.

JavaScript possui outro operador de comparação, `==`, que realiza uma comparação solta. Isso significa que ele pode converter implicitamente os valores dos operandos para o mesmo tipo antes da comparação. Por exemplo, 1 == "1" retorna true porque 1 (número) é convertido para a string "1" para realizar a comparação.

`isNaN()` é uma função global usada para determinar se um valor é Not a Number (Não é um Número). Ela verifica se o valor passado como argumento representa um NaN (Not a Number) especial.

`parseInt()` é uma função usada para analisar uma string e convertê-la em um número inteiro (inteiro). Ela analisa o início da string fornecida e extrai o número inteiro a partir dela, considerando uma base específica (sistema de numeração) opcional. Neste caso seria `parseInt(string, base)`

`continue` e `break` funcionam exatamente como em outras linguagens como C++ ou Python: o primeiro força a reanálise da condição do laço, enquanto o segundo força uma saída incondicional do mesmo.

Finalmente, faltou dizer que numa sessão do browser, se você apertar F12, vai abrir uma janela na qual apertando a aba CONSOLE você terá uma janela para emitir comandos JS e obter eventuais respostas.

☞ Para você fazer

Você deve digitar/copiar este programa, salvando-o no mesmo diretório de trabalho onde vai criar depois o HTML que o chamará.

Use um nome qualquer para este programa, digamos `XX.JS`.

A seguir, escreva um HTML simples como em

```
<html>
<head> <meta charset="UTF-8">
<script src="xx.js"> </script> </head>
</html>
```

Execute o programa e o HTML clicando neste último. Veja o desempenho do programa.

Apenas para aprender mais uma, exclua a cláusula `<meta charset="UTF-8">` e reexecute o programa. O que aconteceu ?

Mostre os arquivos gerados no computador ao professor, OU alternativamente imprima os arquivos pedidos grampeie-os nesta folha e devolva tudo ao professor.

Avaliação:

203-76049 - ga/ a

CEP - UP - UTFPR - PUCPr – 27/05/2024 - 10:43:53.8
Prof Dr P. Kantek (pkantek@gmail.com)
O JS como linguagem desktop VIVXh61a, V: 1.05
76113 NICOLE DA SILVA DE AZEVEDO (76113);
FELIPE EIJI KANO (76032);
LAURA PACHECO DELATTRE (76087);
24FRO203 - 3 entregar ate 11/jun _____ / / _____

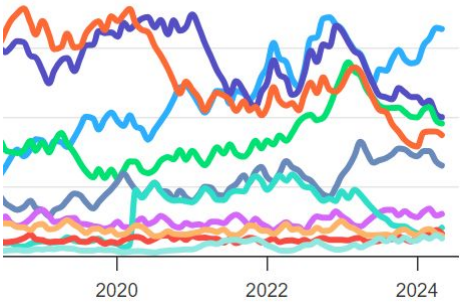
JS como linguagem de desktop

O Javascript tem como vocação principal dar vitalidade a páginas HTML, permitindo que elas ganhem um comportamento dinâmico e sobretudo possam interagir como o usuário da página sendo exibida. Entretanto, graças à tecnologia que floresceu no entorno do javascript, ele acabou se tornando um ambiente de desenvolvimento/produção completo. Esta folha aborda este conceito, que é o de disponibilizar (mais) uma bancada completa para o programador desenvolver sua arte. Relembre que o compilador/interpretador Javascript está dentro do browser e só tem sentido falar em JS ao usar uma sessão desse programa.

Aliás, um parênteses oportuno: o que é melhor:

- Privilegiar uma única linguagem se tornando **o** especialista nela e conhecendo **todos** ou quase todos os meandros e idiossincrasias dela OU
- Conhecer e usar um leque (crescente) de linguagens, plataformas, ambientes diversos, alternando usos, e procurando de cada um o que ele faz bem.

Não tenho resposta para isto, mas a dúvida é propícia para alguns raciocínios. O primeiro enfoque te permite ser um especialista, o que em muitos casos é fortemente buscado pelo mercado de empregos (por exemplo). Além disso, há funcionalidades que só podem ser boladas e desenvolvidas por este personagem. Mas, um problema aqui é a obsolescência meio programada desta expertise. Se você olhar a história recente da informática, verá que inúmeros ambientes já foram a crista da onda e em poucos anos deixaram de sê-lo. Lá atrás, este papel foi assumido pelo Fortran, e depois pelo Cobol e hoje tais linguagem no mínimo, cheiram a naftalina. Mais recentemente o Java pode ser colocado nesse balaio. O queridinho da hora é o Python, mas provavelmente o é até que surja o novo objeto de desejo de todos.



Neste gráfico (TIOBE de maio de 2024) considerado o melhor termômetro de uso de linguagens no mundo, o azul claro é Python, o azul escuro é C, o verde claro é C++ e o laranja é Java. A propósito, o cor de rosa é Javascript.

Já a segunda abordagem nos permite uma maior segurança, já que não coloca todos os ovos na mesma cesta. Mas, não existe almoço grátis, ao transitar por diversos ambientes, há uma dificuldade adicional: a confusão entre eles, e até uma certa complexidade em traduzir um único algoritmo em diversas encarnações. Além do que, é quase impossível ser um completo expert em plataformas diferentes. Mas, de qualquer forma, eu prefiro sempre um poliglota a um monoglota, por mais competente que seja.

Isto posto, vamos olhar o Javascript como um faz-tudo.

Declaração de Variáveis:

var Declara uma variável com escopo de função.

let Declara uma variável com escopo de bloco, ou seja tudo o que está entre { e }.

const Declara uma variável constante (?) com escopo de bloco.

Tipos de Dados:

number Números inteiros e decimais.

string Textos entre aspas simples ou duplas.

boolean Valores True ou False.

array Coleções ordenadas de dados.

object Coleções não ordenadas de dados chave-valor.

Operadores:

Aritméticos (`+`, `-`, `*`, `/`, `%`): Realizam operações matemáticas básicas.

Comparação (`==`, `!=`, `<`, `>`, `<=`, `>=`): Comparação de valores.

Lógicos (`&&`, `||`, `!`): Operações lógicas AND, OR e NOT.

Estruturas de Controle:

if/else Executa um bloco de código se uma condição for verdadeira, senão outro bloco.

switch/case Executa um bloco de código específico com base em uma comparação.

while Repete um bloco de código enquanto uma condição for verdadeira.

for Repete um bloco de código um número específico de vezes.

Funções:

function Cria uma função reutilizável para executar um bloco de código.

return Retorna um valor da função.

Repare que até aqui, não se falou uma única vez de HTML, coerente com o objetivo desta folha.

Ainda falta a entrada e saída, que aqui é meio complicada, afinal não nos esqueçamos que tem um browser no meio da comunicação. Para falar, (output) o programa pode lançar mão de `alert(msg)` que interrompe o programa e lança uma caixa de diálogo (em forma modal). Há também o `var=prompt(msg)` que faz o mesmo mas ainda recupera o que o usuário digitou colocando-o em `var`. O alert interrompe o fluxo do programa que só é retomado após o usuário apertar o botão `OK`. Já o prompt, devolve o que o usuário digitar (se ele apertar `OK`) ou retorna null se ele apertar `Cancelar`).

Veja um exemplo de tudo o que se disse acima. Estude o programa a seguir

Exemplo: Jogo da adivinhação

Este jogo simples desafia o usuário a adivinhar um número secreto gerado aleatoriamente pelo computador. O jogo fornece feedback ao usuário, indicando se o palpite está muito alto, muito baixo ou correto.

```
// Gera um número aleatório entre 1 e 100
let numSecr=Math.floor(Math.random()*100)+1;
// Define o número máximo de tentativas
let tentMax = 10;
// Mantém o controle do número de tentativas
let tentRest = tentMax;
// Loop principal do jogo
while (tentRest > 0) {
  // Obtém o palpite do usuário
  let palpite = prompt(`Digite um número 1..100
  (tentativas restantes: ${tentRest})`);
  // Verifica se o palpite é válido
  if (isNaN(palpite)||palpite<1||palpite>100) {
    alert("Ruim: Digite um número entre 1 e 100.");
    continue;
  }
  // Converte o palpite para um número
  palpite = parseInt(palpite);
  // Verifica se o palpite está correto
  if (palpite === numSecr) {
    alert(`Parabéns! Você adivinhou  em
    ${tentMax - tentRest + 1} tentativas.`);
    break;
  } else if (palpite < numSecr) {
    alert("O número é maior que seu palpite.");
  } else {
    alert("O número é menor que seu palpite.");
  }
  // Decrementa o número de tentativas restantes
  tentRest--;
}
// Informa a mensagem final caso não tenha
adivinhado o número
if (tentRest === 0) {
  alert(`Você não conseguiu adivinhar em
  ${tentMax} tentativas. Ele era ${numSecr}.`);
}
```

Há aqui a construção `{...}` que é conhecida como interpolação de string, é uma ferramenta para formatar strings de forma dinâmica e expressiva. Ela permite incorporar expressões JavaScript dentro de strings literais, resultando em strings personalizadas e dinâmicas.

A construção `${...}` é composta por um par de chaves ({}) que contêm uma expressão JavaScript. A expressão é avaliada e seu valor é inserido na string literal no local onde a construção aparece. Isso permite que você combine strings estáticas com dados dinâmicos gerados a partir de variáveis, funções ou outras operações JavaScript.

o operador === representa a comparação estrita entre dois valores. Ele verifica se os valores dos operandos (elementos que o operador compara) são iguais e do mesmo tipo.

JavaScript possui outro operador de comparação, ==, que realiza uma comparação solta. Isso significa que ele pode converter implicitamente os valores dos operandos para o mesmo tipo antes da comparação. Por exemplo, 1 == ”1” retorna true porque 1 (número) é convertido para a string ”1” para realizar a comparação.

`isNaN()` é uma função global usada para determinar se um valor é Not a Number (Não é um Número). Ela verifica se o valor passado como argumento representa um NaN (Not a Number) especial.

`parseInt()` é uma função usada para analisar uma string e convertê-la em um número inteiro (inteiro). Ela analisa o início da string fornecida e extrai o número inteiro a partir dela, considerando uma base específica (sistema de numeração) opcional. Neste caso seria `parseInt(string, base)`

`continue` e `break` funcionam exatamente como em outras linguagens como C++ ou Python: o primeiro força a reanálise da condição do laço, enquanto o segundo força uma saída incondicional do mesmo.

Finalmente, faltou dizer que numa sessão do browser, se você apertar F12, vai abrir uma janela na qual apertando a aba CONSOLE você terá uma janela para emitir comandos JS e obter eventuais respostas.

☞ Para você fazer

Você deve digitar/copiar este programa, salvando-o no mesmo diretório de trabalho onde vai criar depois o HTML que o chamará.

Use um nome qualquer para este programa, digamos `XX.JS`.

A seguir, escreva um HTML simples como em

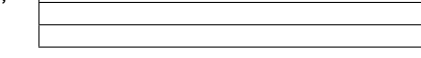
```
<html>
<head> <meta charset="UTF-8">
<script src="xx.js"> </script> </head>
</html>
```

Execute o programa e o HTML clicando neste último. Veja o desempenho do programa.

Apenas para aprender mais uma, exclua a cláusula `<meta charset="UTF-8">` e reexecute o programa. O que aconteceu ?

Mostre os arquivos gerados no computador ao professor, OU alternativamente imprima os arquivos pedidos grampeie-os nesta folha e devolva tudo ao professor.

Avaliação:



203-76113 - ga/ a

CEP - UP - UTFPR - PUCPr – 27/05/2024 - 10:43:53.8
Prof Dr P. Kantek (pkantek@gmail.com)
O JS como linguagem desktop VIVXh61a, V: 1.05
76182 VITOR EDUARDO DE GOES FONTE(76182);
GABRIELA BERNARDO PASQUALOT(76056);
EDUARDO CAVALHEIRO DIAS (76018);
24FRO203 - 4 entregar ate 11/jun _____ / / _____

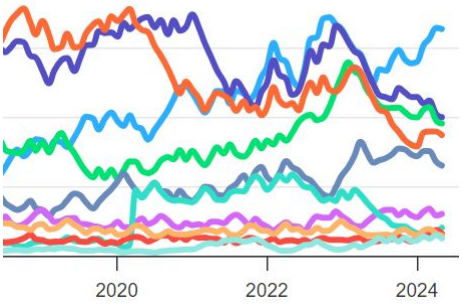
JS como linguagem de desktop

O Javascript tem como vocação principal dar vitalidade a páginas HTML, permitindo que elas ganhem um comportamento dinâmico e sobretudo possam interagir como o usuário da página sendo exibida. Entretanto, graças à tecnologia que floresceu no entorno do javascript, ele acabou se tornando um ambiente de desenvolvimento/produção completo. Esta folha aborda este conceito, que é o de disponibilizar (mais) uma bancada completa para o programador desenvolver sua arte. Relembre que o compilador/interpretador Javascript está dentro do browser e só tem sentido falar em JS ao usar uma sessão desse programa.

Aliás, um parênteses oportuno: o que é melhor:

- Privilegiar uma única linguagem se tornando **o** especialista nela e conhecendo **todos** ou quase todos os meandros e idiossincrasias dela OU
- Conhecer e usar um leque (crescente) de linguagens, plataformas, ambientes diversos, alternando usos, e procurando de cada um o que ele faz bem.

Não tenho resposta para isto, mas a dúvida é propícia para alguns raciocínios. O primeiro enfoque te permite ser um especialista, o que em muitos casos é fortemente buscado pelo mercado de empregos (por exemplo). Além disso, há funcionalidades que só podem ser boladas e desenvolvidas por este personagem. Mas, um problema aqui é a obsolescência meio programada desta expertise. Se você olhar a história recente da informática, verá que inúmeros ambientes já foram a crista da onda e em poucos anos deixaram de sê-lo. Lá atrás, este papel foi assumido pelo Fortran, e depois pelo Cobol e hoje tais linguagem no mínimo, cheiram a naftalina. Mais recentemente o Java pode ser colocado nesse balaio. O queridinho da hora é o Python, mas provavelmente o é até que surja o novo objeto de desejo de todos.



Neste gráfico (TIOBE de maio de 2024) considerado o melhor termômetro de uso de linguagens no mundo, o azul claro é Python, o azul escuro é C, o verde claro é C++ e o laranja é Java. A propósito, o cor de rosa é Javascript.

Já a segunda abordagem nos permite uma maior segurança, já que não coloca todos os ovos na mesma cesta. Mas, não existe almoço grátis, ao transitar por diversos ambientes, há uma dificuldade adicional: a confusão entre eles, e até uma certa complexidade em traduzir um único algoritmo em diversas encarnações. Além do que, é quase impossível ser um completo expert em plataformas diferentes. Mas, de qualquer forma, eu prefiro sempre um poliglota a um monoglota, por mais competente que seja.

Isto posto, vamos olhar o Javascript como um faz-tudo.

Declaração de Variáveis:

var Declara uma variável com escopo de função.

let Declara uma variável com escopo de bloco, ou seja tudo o que está entre { e }.

const Declara uma variável constante (?) com escopo de bloco.

Tipos de Dados:

number Números inteiros e decimais.

string Textos entre aspas simples ou duplas.

boolean Valores True ou False.

array Coleções ordenadas de dados.

object Coleções não ordenadas de dados chave-valor.

Operadores:

Aritméticos (`+`, `-`, `*`, `/`, `%`): Realizam operações matemáticas básicas.

Comparação (`==`, `!=`, `<`, `>`, `<=`, `>=`): Comparação de valores.

Lógicos (`&&`, `||`, `!`): Operações lógicas AND, OR e NOT.

Estruturas de Controle:

if/else Executa um bloco de código se uma condição for verdadeira, senão outro bloco.

switch/case Executa um bloco de código específico com base em uma comparação.

while Repete um bloco de código enquanto uma condição for verdadeira.

for Repete um bloco de código um número específico de vezes.

Funções:

function Cria uma função reutilizável para executar um bloco de código.

return Retorna um valor da função.

Repare que até aqui, não se falou uma única vez de HTML, coerente com o objetivo desta folha.

Ainda falta a entrada e saída, que aqui é meio complicada, afinal não nos esqueçamos que tem um browser no meio da comunicação. Para falar, (output) o programa pode lançar mão de `alert(msg)` que interrompe o programa e lança uma caixa de diálogo (em forma modal). Há também o `var=prompt(msg)` que faz o mesmo mas ainda recupera o que o usuário digitou colocando-o em `var`. O alert interrompe o fluxo do programa que só é retomado após o usuário apertar o botão `OK`. Já o prompt, devolve o que o usuário digitar (se ele apertar `OK`) ou retorna null se ele apertar `Cancelar`).

Veja um exemplo de tudo o que se disse acima. Estude o programa a seguir

Exemplo: Jogo da adivinhação

Este jogo simples desafia o usuário a adivinhar um número secreto gerado aleatoriamente pelo computador. O jogo fornece feedback ao usuário, indicando se o palpite está muito alto, muito baixo ou correto.

```
// Gera um número aleatório entre 1 e 100
let numSecr=Math.floor(Math.random()*100)+1;
// Define o número máximo de tentativas
let tentMax = 10;
// Mantém o controle do número de tentativas
let tentRest = tentMax;
// Loop principal do jogo
while (tentRest > 0) {
  // Obtém o palpite do usuário
  let palpite = prompt(`Digite um número 1..100
  (tentativas restantes: ${tentRest})`);
  // Verifica se o palpite é válido
  if (isNaN(palpite)||palpite<1||palpite>100) {
    alert("Ruim: Digite um número entre 1 e 100.");
    continue;
  }
  // Converte o palpite para um número
  palpite = parseInt(palpite);
  // Verifica se o palpite está correto
  if (palpite === numSecr) {
    alert(`Parabéns! Você adivinhou  em
    ${tentMax - tentRest + 1} tentativas.`);
    break;
  } else if (palpite < numSecr) {
    alert("O número é maior que seu palpite.");
  } else {
    alert("O número é menor que seu palpite.");
  }
  // Decrementa o número de tentativas restantes
  tentRest--;
}
// Informa a mensagem final caso não tenha
adivinhado o número
if (tentRest === 0) {
  alert(`Você não conseguiu adivinhar em
  ${tentMax} tentativas. Ele era ${numSecr}.`);
}
```

Há aqui a construção `{...}` que é conhecida como interpolação de string, é uma ferramenta para formatar strings de forma dinâmica e expressiva. Ela permite incorporar expressões JavaScript dentro de strings literais, resultando em strings personalizadas e dinâmicas.

A construção `${...}` é composta por um par de chaves ({}) que contêm uma expressão JavaScript. A expressão é avaliada e seu valor é inserido na string literal no local onde a construção aparece. Isso permite que você combine strings estáticas com dados dinâmicos gerados a partir de variáveis, funções ou outras operações JavaScript.

o operador === representa a comparação estrita entre dois valores. Ele verifica se os valores dos operandos (elementos que o operador compara) são iguais e do mesmo tipo.

JavaScript possui outro operador de comparação, ==, que realiza uma comparação solta. Isso significa que ele pode converter implicitamente os valores dos operandos para o mesmo tipo antes da comparação. Por exemplo, 1 == ”1” retorna true porque 1 (número) é convertido para a string ”1” para realizar a comparação.

`isNaN()` é uma função global usada para determinar se um valor é Not a Number (Não é um Número). Ela verifica se o valor passado como argumento representa um NaN (Not a Number) especial.

`parseInt()` é uma função usada para analisar uma string e convertê-la em um número inteiro (inteiro). Ela analisa o início da string fornecida e extrai o número inteiro a partir dela, considerando uma base específica (sistema de numeração) opcional. Neste caso seria `parseInt(string, base)`

`continue` e `break` funcionam exatamente como em outras linguagens como C++ ou Python: o primeiro força a reanálise da condição do laço, enquanto o segundo força uma saída incondicional do mesmo.

Finalmente, faltou dizer que numa sessão do browser, se você apertar F12, vai abrir uma janela na qual apertando a aba CONSOLE você terá uma janela para emitir comandos JS e obter eventuais respostas.

☞ Para você fazer

Você deve digitar/copiar este programa, salvando-o no mesmo diretório de trabalho onde vai criar depois o HTML que o chamará.

Use um nome qualquer para este programa, digamos `XX.JS`.

A seguir, escreva um HTML simples como em

```
<html>
<head> <meta charset="UTF-8">
<script src="xx.js"> </script> </head>
</html>
```

Execute o programa e o HTML clicando neste último. Veja o desempenho do programa.

Apenas para aprender mais uma, exclua a cláusula `<meta charset="UTF-8">` e reexecute o programa. O que aconteceu ?

Mostre os arquivos gerados no computador ao professor, OU alternativamente imprima os arquivos pedidos grampeie-os nesta folha e devolva tudo ao professor.

Avaliação:

203-76182 - ga/ a

CEP - UP - UTFPR - PUCPr – 27/05/2024 - 10:43:53.8
Prof Dr P. Kantek (pkantek@gmail.com)
O JS como linguagem desktop VIVXh61a, V: 1.05
76199 DANILO DEPETRIS SOARES (76199);
RAUL KNAPKI CUNHA (76144);
PAULO JOSE FREIRE CORREA (76120);
24FRO203 - 5 entregar ate 11/jun _____ / / _____

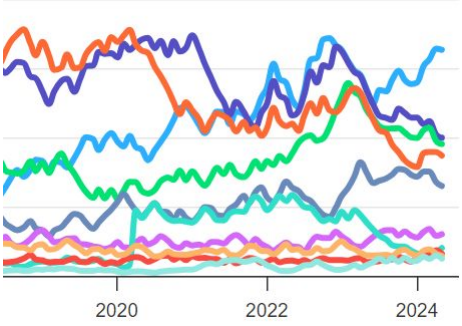
JS como linguagem de desktop

O Javascript tem como vocação principal dar vitalidade a páginas HTML, permitindo que elas ganhem um comportamento dinâmico e sobretudo possam interagir como o usuário da página sendo exibida. Entretanto, graças à tecnologia que floresceu no entorno do javascript, ele acabou se tornando um ambiente de desenvolvimento/produção completo. Esta folha aborda este conceito, que é o de disponibilizar (mais) uma bancada completa para o programador desenvolver sua arte. Relembre que o compilador/interpretador Javascript está dentro do browser e só tem sentido falar em JS ao usar uma sessão desse programa.

Aliás, um parênteses oportuno: o que é melhor:

- Privilegiar uma única linguagem se tornando **o** especialista nela e conhecendo **todos** ou quase todos os meandros e idiossincrasias dela OU
- Conhecer e usar um leque (crescente) de linguagens, plataformas, ambientes diversos, alternando usos, e procurando de cada um o que ele faz bem.

Não tenho resposta para isto, mas a dúvida é propícia para alguns raciocínios. O primeiro enfoque te permite ser um especialista, o que em muitos casos é fortemente buscado pelo mercado de empregos (por exemplo). Além disso, há funcionalidades que só podem ser boladas e desenvolvidas por este personagem. Mas, um problema aqui é a obsolescência meio programada desta expertise. Se você olhar a história recente da informática, verá que inúmeros ambientes já foram a crista da onda e em poucos anos deixaram de sê-lo. Lá atrás, este papel foi assumido pelo Fortran, e depois pelo Cobol e hoje tais linguagem no mínimo, cheiram a naftalina. Mais recentemente o Java pode ser colocado nesse balaio. O queridinho da hora é o Python, mas provavelmente o é até que surja o novo objeto de desejo de todos.



Neste gráfico (TIOBE de maio de 2024) considerado o melhor termômetro de uso de linguagens no mundo, o azul claro é Python, o azul escuro é C, o verde claro é C++ e o laranja é Java. A propósito, o cor de rosa é Javascript.

Já a segunda abordagem nos permite uma maior segurança, já que não coloca todos os ovos na mesma cesta. Mas, não existe almoço grátis, ao transitar por diversos ambientes, há uma dificuldade adicional: a confusão entre eles, e até uma certa complexidade em traduzir um único algoritmo em diversas encarnações. Além do que, é quase impossível ser um completo expert em plataformas diferentes. Mas, de qualquer forma, eu prefiro sempre um poliglota a um monoglota, por mais competente que seja.

Isto posto, vamos olhar o Javascript como um faz-tudo.

Declaração de Variáveis:

var Declara uma variável com escopo de função.

let Declara uma variável com escopo de bloco, ou seja tudo o que está entre { e }.

const Declara uma variável constante (?) com escopo de bloco.

Tipos de Dados:

number Números inteiros e decimais.

string Textos entre aspas simples ou duplas.

boolean Valores True ou False.

array Coleções ordenadas de dados.

object Coleções não ordenadas de dados chave-valor.

Operadores:

Aritméticos (`+`, `-`, `*`, `/`, `%`): Realizam operações matemáticas básicas.

Comparação (`==`, `!=`, `<`, `>`, `<=`, `>=`): Comparação de valores.

Lógicos (`&&`, `||`, `!`): Operações lógicas AND, OR e NOT.

Estruturas de Controle:

if/else Executa um bloco de código se uma condição for verdadeira, senão outro bloco.

switch/case Executa um bloco de código específico com base em uma comparação.

while Repete um bloco de código enquanto uma condição for verdadeira.

for Repete um bloco de código um número específico de vezes.

Funções:

function Cria uma função reutilizável para executar um bloco de código.

return Retorna um valor da função.

Repare que até aqui, não se falou uma única vez de HTML, coerente com o objetivo desta folha.

Ainda falta a entrada e saída, que aqui é meio complicada, afinal não nos esqueçamos que tem um browser no meio da comunicação. Para falar, (output) o programa pode lançar mão de `alert(msg)` que interrompe o programa e lança uma caixa de diálogo (em forma modal). Há também o `var=prompt(msg)` que faz o mesmo mas ainda recupera o que o usuário digitou colocando-o em `var`. O alert interrompe o fluxo do programa que só é retomado após o usuário apertar o botão `OK`. Já o prompt, devolve o que o usuário digitar (se ele apertar `OK`) ou retorna null se ele apertar `Cancelar`).

Veja um exemplo de tudo o que se disse acima. Estude o programa a seguir

Exemplo: Jogo da adivinhação

Este jogo simples desafia o usuário a adivinhar um número secreto gerado aleatoriamente pelo computador. O jogo fornece feedback ao usuário, indicando se o palpite está muito alto, muito baixo ou correto.

```
// Gera um número aleatório entre 1 e 100
let numSecr=Math.floor(Math.random()*100)+1;
// Define o número máximo de tentativas
let tentMax = 10;
// Mantém o controle do número de tentativas
let tentRest = tentMax;
// Loop principal do jogo
while (tentRest > 0) {
  // Obtém o palpite do usuário
  let palpite = prompt(`Digite um número 1..100
  (tentativas restantes: ${tentRest})`);
  // Verifica se o palpite é válido
  if (isNaN(palpite)||palpite<1||palpite>100) {
    alert("Ruim: Digite um número entre 1 e 100.");
    continue;
  }
  // Converte o palpite para um número
  palpite = parseInt(palpite);
  // Verifica se o palpite está correto
  if (palpite === numSecr) {
    alert(`Parabéns! Você adivinhou  em
    ${tentMax - tentRest + 1} tentativas.`);
    break;
  } else if (palpite < numSecr) {
    alert("O número é maior que seu palpite.");
  } else {
    alert("O número é menor que seu palpite.");
  }
  // Decrementa o número de tentativas restantes
  tentRest--;
}
// Informa a mensagem final caso não tenha
adivinhado o número
if (tentRest === 0) {
  alert(`Você não conseguiu adivinhar em
  ${tentMax} tentativas. Ele era ${numSecr}.`);
}
```

Há aqui a construção `{...}` que é conhecida como interpolação de string, é uma ferramenta para formatar strings de forma dinâmica e expressiva. Ela permite incorporar expressões JavaScript dentro de strings literais, resultando em strings personalizadas e dinâmicas.

A construção `${...}` é composta por um par de chaves ({}) que contêm uma expressão JavaScript. A expressão é avaliada e seu valor é inserido na string literal no local onde a construção aparece. Isso permite que você combine strings estáticas com dados dinâmicos gerados a partir de variáveis, funções ou outras operações JavaScript.

o operador `===` representa a comparação estrita entre dois valores. Ele verifica se os valores dos operandos (elementos que o operador compara) são iguais e do mesmo tipo.

JavaScript possui outro operador de comparação, `==`, que realiza uma comparação solta. Isso significa que ele pode converter implicitamente os valores dos operandos para o mesmo tipo antes da comparação. Por exemplo, 1 == "1" retorna true porque 1 (número) é convertido para a string "1" para realizar a comparação.

`isNaN()` é uma função global usada para determinar se um valor é Not a Number (Não é um Número). Ela verifica se o valor passado como argumento representa um NaN (Not a Number) especial.

`parseInt()` é uma função usada para analisar uma string e convertê-la em um número inteiro (inteiro). Ela analisa o início da string fornecida e extrai o número inteiro a partir dela, considerando uma base específica (sistema de numeração) opcional. Neste caso seria `parseInt(string, base)`

`continue` e `break` funcionam exatamente como em outras linguagens como C++ ou Python: o primeiro força a reanálise da condição do laço, enquanto o segundo força uma saída incondicional do mesmo.

Finalmente, faltou dizer que numa sessão do browser, se você apertar F12, vai abrir uma janela na qual apertando a aba CONSOLE você terá uma janela para emitir comandos JS e obter eventuais respostas.

☞ Para você fazer

Você deve digitar/copiar este programa, salvando-o no mesmo diretório de trabalho onde vai criar depois o HTML que o chamará.

Use um nome qualquer para este programa, digamos `XX.JS`.

A seguir, escreva um HTML simples como em

```
<html>
<head> <meta charset="UTF-8">
<script src="xx.js"> </script> </head>
</html>
```

Execute o programa e o HTML clicando neste último. Veja o desempenho do programa.

Apenas para aprender mais uma, exclua a cláusula `<meta charset="UTF-8">` e reexecute o programa. O que aconteceu ?

Mostre os arquivos gerados no computador ao professor, OU alternativamente imprima os arquivos pedidos grampeie-os nesta folha e devolva tudo ao professor.

Avaliação:

203-76199 - ga/ a

CEP - UP - UTFPR - PUCPr – 27/05/2024 - 10:43:53.8
Prof Dr P. Kantek (pkantek@gmail.com)
O JS como linguagem desktop VIVXh61a, V: 1.05
76175 VICTOR HUGO DOS SANTOS DE C(76175);
CARLOS EDUARDO ANTAO (76001);
24FRO203 - 6 entregar ate 11/jun _____ / / _____

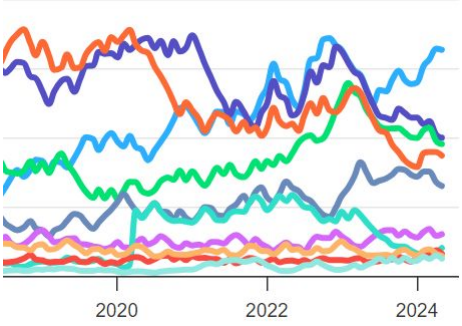
JS como linguagem de desktop

O Javascript tem como vocação principal dar vitalidade a páginas HTML, permitindo que elas ganhem um comportamento dinâmico e sobretudo possam interagir como o usuário da página sendo exibida. Entretanto, graças à tecnologia que floresceu no entorno do javascript, ele acabou se tornando um ambiente de desenvolvimento/produção completo. Esta folha aborda este conceito, que é o de disponibilizar (mais) uma bancada completa para o programador desenvolver sua arte. Relembre que o compilador/interpretador Javascript está dentro do browser e só tem sentido falar em JS ao usar uma sessão desse programa.

Aliás, um parênteses oportuno: o que é melhor:

- Privilegiar uma única linguagem se tornando **o** especialista nela e conhecendo **todos** ou quase todos os meandros e idiossincrasias dela OU
- Conhecer e usar um leque (crescente) de linguagens, plataformas, ambientes diversos, alternando usos, e procurando de cada um o que ele faz bem.

Não tenho resposta para isto, mas a dúvida é propícia para alguns raciocínios. O primeiro enfoque te permite ser um especialista, o que em muitos casos é fortemente buscado pelo mercado de empregos (por exemplo). Além disso, há funcionalidades que só podem ser boladas e desenvolvidas por este personagem. Mas, um problema aqui é a obsolescência meio programada desta expertise. Se você olhar a história recente da informática, verá que inúmeros ambientes já foram a crista da onda e em poucos anos deixaram de sê-lo. Lá atrás, este papel foi assumido pelo Fortran, e depois pelo Cobol e hoje tais linguagem no mínimo, cheiram a naftalina. Mais recentemente o Java pode ser colocado nesse balaio. O queridinho da hora é o Python, mas provavelmente o é até que surja o novo objeto de desejo de todos.



Neste gráfico (TIOBE de maio de 2024) considerado o melhor termômetro de uso de linguagens no mundo, o azul claro é Python, o azul escuro é C, o verde claro é C++ e o laranja é Java. A propósito, o cor de rosa é Javascript.

Já a segunda abordagem nos permite uma maior segurança, já que não coloca todos os ovos na mesma cesta. Mas, não existe almoço grátis, ao transitar por diversos ambientes, há uma dificuldade adicional: a confusão entre eles, e até uma certa complexidade em traduzir um único algoritmo em diversas encarnações. Além do que, é quase impossível ser um completo expert em plataformas diferentes. Mas, de qualquer forma, eu prefiro sempre um poliglota a um monoglota, por mais competente que seja.

Isto posto, vamos olhar o Javascript como um faz-tudo.

Declaração de Variáveis:

var Declara uma variável com escopo de função.

let Declara uma variável com escopo de bloco, ou seja tudo o que está entre { e }.

const Declara uma variável constante (?) com escopo de bloco.

Tipos de Dados:

number Números inteiros e decimais.

string Textos entre aspas simples ou duplas.

boolean Valores True ou False.

array Coleções ordenadas de dados.

object Coleções não ordenadas de dados chave-valor.

Operadores:

Aritméticos (`+`, `-`, `*`, `/`, `%`): Realizam operações matemáticas básicas.

Comparação (`==`, `!=`, `<`, `>`, `<=`, `>=`): Comparação de valores.

Lógicos (`&&`, `||`, `!`): Operações lógicas AND, OR e NOT.

Estruturas de Controle:

if/else Executa um bloco de código se uma condição for verdadeira, senão outro bloco.

switch/case Executa um bloco de código específico com base em uma comparação.

while Repete um bloco de código enquanto uma condição for verdadeira.

for Repete um bloco de código um número específico de vezes.

Funções:

function Cria uma função reutilizável para executar um bloco de código.

return Retorna um valor da função.

Repare que até aqui, não se falou uma única vez de HTML, coerente com o objetivo desta folha.

Ainda falta a entrada e saída, que aqui é meio complicada, afinal não nos esqueçamos que tem um browser no meio da comunicação. Para falar, (output) o programa pode lançar mão de `alert(msg)` que interrompe o programa e lança uma caixa de diálogo (em forma modal). Há também o `var=prompt(msg)` que faz o mesmo mas ainda recupera o que o usuário digitou colocando-o em `var`. O alert interrompe o fluxo do programa que só é retomado após o usuário apertar o botão `OK`. Já o prompt, devolve o que o usuário digitar (se ele apertar `OK`) ou retorna null se ele apertar `Cancelar`).

Veja um exemplo de tudo o que se disse acima. Estude o programa a seguir

Exemplo: Jogo da adivinhação

Este jogo simples desafia o usuário a adivinhar um número secreto gerado aleatoriamente pelo computador. O jogo fornece feedback ao usuário, indicando se o palpite está muito alto, muito baixo ou correto.

```
// Gera um número aleatório entre 1 e 100
let numSecr=Math.floor(Math.random()*100)+1;
// Define o número máximo de tentativas
let tentMax = 10;
// Mantém o controle do número de tentativas
let tentRest = tentMax;
// Loop principal do jogo
while (tentRest > 0) {
  // Obtém o palpite do usuário
  let palpite = prompt(`Digite um número 1..100
  (tentativas restantes: ${tentRest})`);
  // Verifica se o palpite é válido
  if (isNaN(palpite)||palpite<1||palpite>100) {
    alert("Ruim: Digite um número entre 1 e 100.");
    continue;
  }
  // Converte o palpite para um número
  palpite = parseInt(palpite);
  // Verifica se o palpite está correto
  if (palpite === numSecr) {
    alert(`Parabéns! Você adivinhou  em
    ${tentMax - tentRest + 1} tentativas.`);
    break;
  } else if (palpite < numSecr) {
    alert("O número é maior que seu palpite.");
  } else {
    alert("O número é menor que seu palpite.");
  }
  // Decrementa o número de tentativas restantes
  tentRest--;
}
// Informa a mensagem final caso não tenha
adivinhado o número
if (tentRest === 0) {
  alert(`Você não conseguiu adivinhar em
  ${tentMax} tentativas. Ele era ${numSecr}.`);
}
```

Há aqui a construção `{...}` que é conhecida como interpolação de string, é uma ferramenta para formatar strings de forma dinâmica e expressiva. Ela permite incorporar expressões JavaScript dentro de strings literais, resultando em strings personalizadas e dinâmicas.

A construção `${...}` é composta por um par de chaves ({}) que contêm uma expressão JavaScript. A expressão é avaliada e seu valor é inserido na string literal no local onde a construção aparece. Isso permite que você combine strings estáticas com dados dinâmicos gerados a partir de variáveis, funções ou outras operações JavaScript.

o operador `===` representa a comparação estrita entre dois valores. Ele verifica se os valores dos operandos (elementos que o operador compara) são iguais e do mesmo tipo.

JavaScript possui outro operador de comparação, `==`, que realiza uma comparação solta. Isso significa que ele pode converter implicitamente os valores dos operandos para o mesmo tipo antes da comparação. Por exemplo, 1 == "1" retorna true porque 1 (número) é convertido para a string "1" para realizar a comparação.

`isNaN()` é uma função global usada para determinar se um valor é Not a Number (Não é um Número). Ela verifica se o valor passado como argumento representa um NaN (Not a Number) especial.

`parseInt()` é uma função usada para analisar uma string e convertê-la em um número inteiro (inteiro). Ela analisa o início da string fornecida e extrai o número inteiro a partir dela, considerando uma base específica (sistema de numeração) opcional. Neste caso seria `parseInt(string, base)`

`continue` e `break` funcionam exatamente como em outras linguagens como C++ ou Python: o primeiro força a reanálise da condição do laço, enquanto o segundo força uma saída incondicional do mesmo.

Finalmente, faltou dizer que numa sessão do browser, se você apertar F12, vai abrir uma janela na qual apertando a aba CONSOLE você terá uma janela para emitir comandos JS e obter eventuais respostas.

☞ Para você fazer

Você deve digitar/copiar este programa, salvando-o no mesmo diretório de trabalho onde vai criar depois o HTML que o chamará.

Use um nome qualquer para este programa, digamos `XX.JS`.

A seguir, escreva um HTML simples como em

```
<html>
<head> <meta charset="UTF-8">
<script src="xx.js"> </script> </head>
</html>
```

Execute o programa e o HTML clicando neste último. Veja o desempenho do programa.

Apenas para aprender mais uma, exclua a cláusula `<meta charset="UTF-8">` e reexecute o programa. O que aconteceu ?

Mostre os arquivos gerados no computador ao professor, OU alternativamente imprima os arquivos pedidos grampeie-os nesta folha e devolva tudo ao professor.

Avaliação:

203-76175 - ga/ a