**Virkņu (string) metodes**

**Virkņu apvienošana**

Lai savienotu virknes, vari izmantot metodi **concat()**. Tā nemaina sākotnējās virknes, bet atgriež savienošanas rezultātu kā virkni. Rezultāts ir jāsaglabā jaunā mainīgajā, pretējā gadījumā tas tiks zaudēts:

|  |  |
| --- | --- |
| Attēls, kurā ir teksts, fonts, ekrānuzņēmums  Mākslīgā intelekta ģenerēts saturs var būt nepareizs. | ***🡨Izmēģini pats!*** |
|  |  |

**Virknes konvertēšana par datu masīvu**

Virkni var pārvērst masīvā, izmantojot metodi **split()**. Atkal ir jāfiksē rezultāts, jo šī metode nemaina sākotnējās virknes. Ņem iepriekšējā piemēra rezultātu: **Sveiks Edvard**! Norādi metodei **split()**, pie kuras vērtības tai vajadzētu sadalīt datus. Katru reizi, kad tā sastopas ar šo vērtību, tā izveidos jaunu datu masīva elementu:

|  |  |
| --- | --- |
| Attēls, kurā ir teksts, ekrānuzņēmums, fonts  Mākslīgā intelekta ģenerēts saturs var būt nepareizs. | ***🡨Izmēģini pats!*** |
| Attēls, kurā ir teksts, fonts, ekrānuzņēmums, cipars  Mākslīgā intelekta ģenerēts saturs var būt nepareizs. |  |

Kā redzi, metode izveido visu elementu masīvu, kas atdalīti ar atstarpi. Elementus var atdalīt ar jebkuru rakstzīmi, piemēram, komatu:

|  |  |  |
| --- | --- | --- |
|  | | ***🡨Izmēģini pats!*** |
| Attēls, kurā ir teksts, ekrānuzņēmums, fonts, cipars  Mākslīgā intelekta ģenerēts saturs var būt nepareizs. | 🡨Skripts ir izveidojis datu masīvu ar sešiem elementiem. | |

**Masīva konvertēšana virknē**

Masīvu var pārvērst virknē, izmantojot metodi **join()**. Lūk, te tas ir parādīts:

|  |  |
| --- | --- |
|  | ***🡨Izmēģini pats!*** |
| Attēls, kurā ir teksts, ekrānuzņēmums, fonts, cipars  Mākslīgā intelekta ģenerēts saturs var būt nepareizs. |  |

Ja vēlaties komata vietā izmantot kaut ko citu, norādiet to šādi:

|  |  |
| --- | --- |
|  | ***🡨Izmēģini pats!*** |
|  |  |

**Darbs ar īpašībām index un position**

Iespēja uzzināt konkrētas apakšvirknes indeksa vērtību virknē ir ļoti noderīga. Piemēram, ja meklējat konkrētu vārdu failā ar ierakstītu lietotāja ievadi, lai izveidotu apakšvirkni, sākot ar šo indeksu. Lūk, kā atrast virknes indeksu. **indexOf()** metode atgriezīs vienu vērtību — apakšvirknes pirmās rakstzīmes indeksu:

|  |  |
| --- | --- |
|  | ***🡨Izmēģini pats!*** |
|  |  |

Rezultāts ir 7. Tas ir tāpēc, ka pirmā “**re**” vērtība parādās vārdā “**are**”, un pirmās rakstzīmes indekss šajā virknē ir ar numuru 7.

Ja indekss netiek atrasts, tiks atgriezta vērtība **-1**. Testē!

|  |  |
| --- | --- |
|  | ***🡨Izmēģini pats!*** |
|  |  |

Alternatīvs veids, kā virknē atrast konkrētu apakšvirkni, ir izmantot metodi **search()**:

|  |  |
| --- | --- |
|  | ***🡨Izmēģini pats!*** |
|  |  |

Rezultāts būs 17, kas pirmās rakstzīmes indekss vārdā "fellow". Ja vērtība netiek atrasta, **search()** metode atgriezīs -1.

Funkcija **search()** kā ievadi izmantos regulāras izteiksmes formātu, savukārt **indexOf()** vienkārši izmanto virkni. Funkcija **indexOf()** ir ātrāka nekā **search()**, bet, ja vēlaties meklēt tikai virkni, izmantojiet **indexOf()**. Ja vēlaties virknes modeli, izmantojiet metodi **search()**.

|  |
| --- |
| **Regulārā izteiksme** ir īpaša sintakse virkņu modeļu definēšanai, ko var izmantot, lai aizstātu visas meklēšanas laikā radušās vērtības. |

Kā atceries, **indexOf()** metode atgriež pirmās atrastās vērtības indeksu. Pastāv arī līdzīga metode **lastIndexOf()**, kas atgriež tās pozīcijas indeksu, kurā virknes arguments tiek sastapts pēdējo reizi. Ja **lastIndex()** neatrod atbilstības, tad izvada -1. Piemēram:

|  |  |
| --- | --- |
|  | ***🡨Izmēģini pats!*** |
|  |  |

Šī darbība atgriež vērtību **24**. Šī ir pēdējā reize, kad vārds "**re**" parādās dzejolī — otrajā vārdā "**are**".

Dažreiz būs jādara pretēji: nevis jāmeklē indekss, zem kura atrodas virkne, bet gan jānoskaidro, kura rakstzīme atrodas zem noteikta indeksa. Šeit noder **charAt(index)** metode, kas kā argumentu ņem norādīto indeksa pozīciju:

|  |  |
| --- | --- |
|  | ***🡨Izmēģini pats!*** |
|  |  |

Konsole izdrukā vērtību **r**, jo rakstzīme indeksā 10 ir vārda "**red**" rakstzīme "**r**". Ja vaicāsi indeksa pozīciju, kas atrodas ārpus virknes diapazona, tā atgriezīs tukšu virkni. Ja vaicāsi mainīgā **pozicija** tipu, saņemsi vērtību **string**.

|  |  |
| --- | --- |
|  | ***🡨Izmēģini pats!*** |
| Tukša rinda |  |

**Apakšvirkņu izveide**

Apakšvirknes var izveidot, izmantojot metodi **slice(start, end)**. Tā nemaina sākotnējo virkni, bet atgriež jaunu virkni, kas sastāv no atlasītās apakšvirknes vērtībām. Metode pieņem divus parametrus: pirmo indeksu, no kura metode sāksies, un otro - beigu indeksu. Ja izlaidīsiet otro indeksu, metode tiks izpildīta no sākuma indeksa līdz virknes beigām. Beigu indekss netiek iekļauts apakšvirknē. Šeit ir piemērs:

|  |  |
| --- | --- |
|  | ***🡨Izmēģini pats!*** |
|  |  |

turpinājums

Pirmajā apakšvirknē ir tikai viens arguments, tāpēc tā sākas ar **indeksu 5** (kurā ir vērtība **e**) un iekļauj visas virknes vērtības **līdz beigām**. Otrajā apakšvirknē ir divi argumenti: **0** un **3** (**C** indekss ir **0** un **a** indekss **3**). Tā kā pēdējais indekss nav iekļauts apakšvirknē, metode atgriezīs tikai **Cre**.

**Virknes fragmentu aizstāšana**

Metode **replace(old, new)** tiek izmantota, lai aizstātu virknes daļu. Tā saņem divus argumentus. Pirmais arguments ir virknes mainīgais, kas jāatrod virknē. Otrais arguments ir jaunā vērtība, kas jāaizstāj. Šeit ir piemērs:

|  |  |
| --- | --- |
|  | ***🡨Izmēģini pats!*** |
|  |  |

Tiks parādīts ziņojums “Sveiks, Pascal!”. Ja mainīgais, uz kuru orientējaties, sākotnējā virknē nav atrodams, aizstāšana netiek veikta un tiek atgriezta sākotnējā virkne, piemēram:

|  |  |
| --- | --- |
|  | ***🡨Izmēģini pats!*** |
|  |  |

**Vēl viena piezīme!!!:** pēc noklusējuma tiek modificēts tikai pirmais atbilstības gadījums. Tāpēc šajā piemērā tiks aizstāts tikai pirmais ieraksts:

|  |  |
| --- | --- |
|  | ***🡨Izmēģini pats!*** |
|  |  |

Ja vēlies aizstāt visas vērtības, izmantojiet metodi **replaceAll()**:

|  |  |
| --- | --- |
|  | ***🡨Izmēģini pats!*** |
|  |  |

**AUGŠĒJAIS REĢISTRS un apakšējais reģistrs**

Virknes burtus var mainīt, izmantojot **toUpperCase()** un **toLowerCase()** — iebūvētas metodes darbam ar virkņu mainīgajiem. Atkal ir jāfiksē rezultāts (šī metode nemaina sākotnējo virkni). Metode **toUpperCase()** pārveido rakstzīmes augšējā reģistrā:

|  |  |
| --- | --- |
| Attēls, kurā ir teksts, fonts, ekrānuzņēmums  Mākslīgā intelekta ģenerēts saturs var būt nepareizs. | ***🡨Izmēģini pats!*** |
|  |  |

Metode **toLowerCase()** pārveido rakstzīmes apakšējā reģistrā:

|  |  |
| --- | --- |
|  | ***🡨Izmēģini pats!*** |
|  |  |

Padarīsim uzdevumu nedaudz sarežģītāku un norādīsim, ka teikuma pirmais vārds jāraksta ar lielo burtu. To var izdarīt, apvienojot dažas no metodēm:

|  |  |
| --- | --- |
|  | ***🡨Izmēģini pats!*** |
|  |  |

Augšminētajā piemērā ir apvienotas vairākas metodes:

|  |  |
| --- | --- |
| **charAt(0)** | pirmā rakstzīme “**s**” |
| **toUpperCase()** | tiek pārveidota augšējā reģistrā “**S**” |
| **concat()** | burts “**S**” tiek savienots (concat) ar atlikušo virknes daļu |
| **slice()** | tiek iegūta atlikušā virknes daļa: “**veiks, kā klājas?**” |

**Virknes sākums un beigas**

Dažreiz ir jāzina, kur sākas un beidzas virkne. Piemērs:

|  |  |
| --- | --- |
| Attēls, kurā ir teksts, ekrānuzņēmums, fonts  Mākslīgā intelekta ģenerēts saturs var būt nepareizs. | ***🡨Izmēģini pats!*** |
|  |  |

Ekrānā tiks parādīta vērtība “**true**”, kas norāda, ka teikums patiešām sākas ar vārdu “**Tev**”. Ņem vērā, ka metode ir reģistrjutīga, tāpēc šāds kods atgriezīs vērtību “**false**”:

|  |  |
| --- | --- |
|  | ***🡨Izmēģini pats!*** |
|  |  |

Lai noteiktu virknes beigas:

|  |  |
| --- | --- |
|  | ***🡨Izmēģini pats!*** |
|  |  |

**Uzdevums “Lielie sākuma burti”**

Izveido funkciju, kas atgriež virkni, kurā katrs vārds ir rakstīts ar lielo burtu. Piemēram, tiek ievadīts teikums "kaTrs vārds tiks rakStīts ar lielo burtu" un tas tiek pārveidots par: "Katrs Vārds Tiks Rakstīts Ar Lielo Burtu".

Turpinājums2

**Skaitliskās metodes**

**Pārbaude – vai tas ir skaitlis?**

Šo darbību var veikt, izmantojot **isNaN()**.

**isNaN()** atgriež vērtību **true**, ja vērtība nav skaitliska. Tomēr bieži vien būs nepieciešams apgrieztais rezultāts. To var iegūt, izmantojot prefiksu **!**:

|  |  |
| --- | --- |
|  | ***🡨Izmēģini pats!*** |
|  |  |

Tā kā **x** ir **skaitlis**, isNaN tiks novērtēts kā **false**. Taču vēlāk šis rezultāts mainīsies uz pretējo — **true**, jo kodā mēs izmantojām prefiksu **!**. Virkne **Čau!** nav skaitliska vērtība, tāpēc konsolē parādīsies vērtība **true**.

**Apskati šo situāciju:**

|  |  |
| --- | --- |
|  |  |

Šeit notiek kaut kas dīvains. Lai gan **5** ir **pēdiņās**, JavaScript to joprojām uztver kā skaitli un atgriež vērtību “**false**”. JavaScript ir ļoti saprotošs! ☺ Kaut visi būtu tikpat saprotoši kā JavaScript ☺.

**Lielākā un mazākā skaitļa atrašana**

Lai atrastu lielāko skaitli no argumentiem, ir iebūvēta metode **Math.max()**. Apskati to darbībā:

|  |  |
| --- | --- |
|  |  |

Lai atrastu mazāko skaitli, lieto metodi **Math.min()**:

|  |  |
| --- | --- |
|  |  |

Ja mēģināsiet šīs metodes pielietot neskaitliskiem argumentiem, rezultāts būs **NaN**:

|  |  |
| --- | --- |
|  |  |

Metode neizvada vērtību 233. Tā analizē visas vērtības un, tā kā nevar noteikt, vai **Čau** ir lielāks vai mazāks par 56, atgriež **NaN**.

**Kvadrātsakne un kāpināšana**

Lai aprēķinātu skaitļa kvadrātsakni, tiek izmantota **Math.sqrt()** metode.

|  |  |
| --- | --- |
|  |  |

Lai skaitli kāpinātu noteiktā pakāpē lieto metodi **Math.pow()**:

|  |  |
| --- | --- |
|  |  |

**Noapaļošana**

Metode **Math.round()**

|  |  |
| --- | --- |
|  |  |

Kā redzams, šeit tiek veikta standarta noapaļošana. Taču ir gadījumi, kad vērtība ir jānoapaļo uz augšu. Piemēram, ja tiek aprēķināts nepieciešamais dēļu skaits un tiek iegūsts rezultāts 1,1. Viens dēlis acīmredzami nav pietiekams, lai pabeigtu darbu. Būs nepieciešami divi. Šajā gadījumā ieteicams izmantot **Math.ceil()** metodi:

|  |  |
| --- | --- |
|  |  |

**Math.ceil()** metode vienmēr noapaļo skaitli uz augšu līdz tuvākajam veselajam skaitlim. Taču esi uzmanīgs ar negatīviem skaitļiem, jo ​​-5 ir lielāks par -6. Kā tas darbojas, varat redzēt zemāk:

|  |  |
| --- | --- |
|  |  |

**Math.floor()** metode ir pilnīgs pretstats **Math.ceil()** metodei. Tā noapaļo skaitli uz leju līdz tuvākajam veselajam skaitlim.

|  |  |
| --- | --- |
|  |  |

Esiet uzmanīgi ar negatīviem skaitļiem:

|  |  |
| --- | --- |
|  |  |

Metode **Math.trunc()** pozitīviem skaitļiem sniedz tieši tādu pašu rezultātu kā **Math.floor()**, taču rezultātu iegūst citādā veidā. Būtībā tā nenoapaļo uz leju, bet vienkārši atgriež veselo skaitli:

|  |  |
| --- | --- |
|  |  |

Atšķirību var redzēt, ja metodi **Math.trunc()**piemērojam negatīviem skaitļiem:

|  |  |
| --- | --- |
|  |  |