Tell me about yourself?

Thanks for giving me this golden opportunity to introduce myself before you.

My name is Ganesh Pataiya I am from Pune. I have done my master degree from RGTU University Bhopal. I love to work on user interfaces (UI). I started my journey in print media as a freelancer graphics designer. I had made up my mind to become a UI UX designer/Developer. After that I got chance to work with many reputed IT firm like Milestone Internet marketing, Vincent IT, IBS as a UI designer. Currently working with Effective Digital there product called "GreenOrbit" as a UI UX developer.

My Role in current Company:

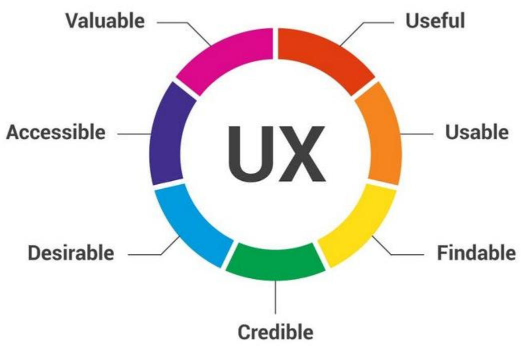
Develop UI using HTML and CSS/SASS/JS or other style sheet languages.

- Source control and Jira
- Testing
- Code reviews
- Communication
- Training and Mentoring

What is User Experience (UX) Design?

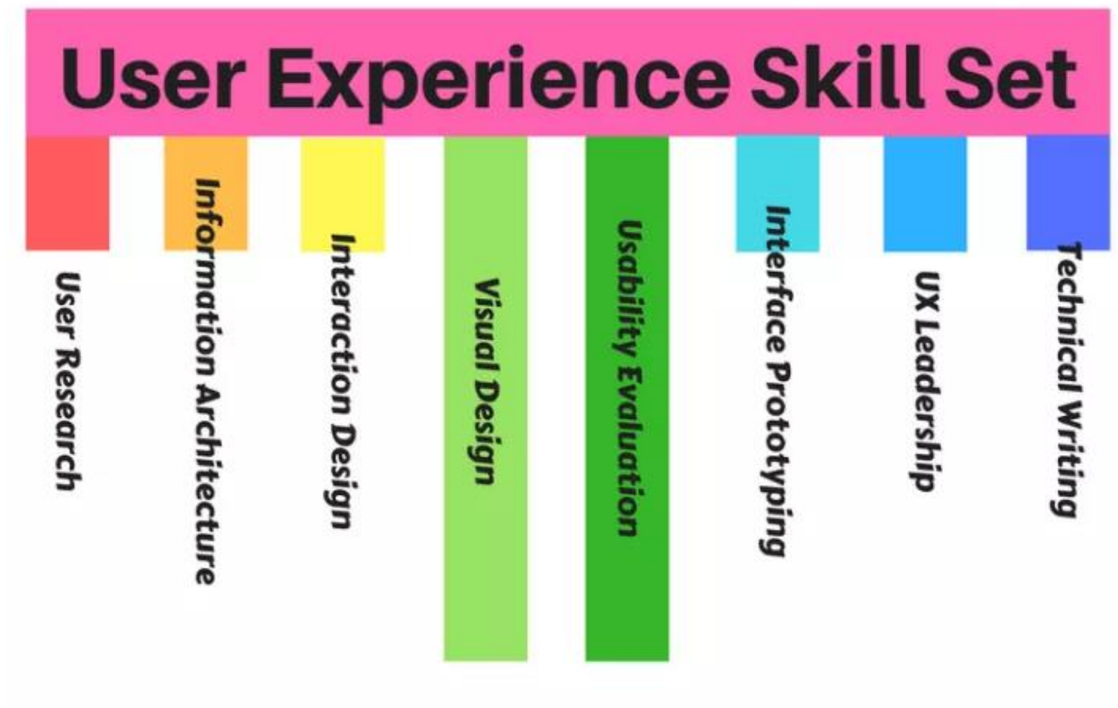
User experience (UX) design is the process of creating products that provide meaningful and relevant experiences to users.

This involves the design of the entire process of acquiring and integrating the product, including aspects of branding, design, usability, and function.



User interface design requires a good understanding of user needs. There are several phases and processes in the user interface design, some of which are more demanded upon than others, depending on the project.[2] (Note: for the remainder of this section, the word *system* is used to denote any project whether it is a website, application, or device.)

Ref. https://blog.usertest.io/what-makes-a-good-ux-designer/



What Technical Skills Does A UX Designer Need?

UX technical skills can be classified into 8 categories:

- User Research
- Information Architecture

- Interaction Design
- Visual Design
- Usability Evaluation
- Interface Prototyping
- UX Leadership
- Technical Writing

However, due to team dynamics and specific requirements/demands by businesses, UX roles cannot be judged purely based on the accumulation of varied skills and how proficient the individual is.

Nonetheless, let's explore each skill area to understand why these skills are important to UX and why a UX designer may need them.

User Research

Ref. https://www.youtube.com/watch?v=TMjCfcHz23A

Contextual Enquiry

- Beginning of the design process
- At user's Natural Environment

Interview

- Face to face interaction
- Before and after Product Design
- User needs & feelings

Diary Studies

- Tracking user Behaviour
- Understand user’s user Pattern

User Group

- Group of Targeted Audience
- Set of Topics for Discussions

Usability Testing

- Perform in lab with session recording
- Evaluating product by performing task

Survey/Questionnaire

- Quantitative & Qualitative Questions
- Feedback’s & Suggestions

Card Sorting

- Open & Close Ended Questions
- Architecturing & Categorizing Information

UX principles and full UX stack

The goals on each level of the UX design pyramid could be reached through following the main **UX design principles**:

1. **Hierarchy**
   Hierarchy is one of the designers' best tools to help users move through a product easily. It includes:
   a). Information Architecture (the way how content is organized across the app or site) and a
   b). Visual Hierarchy (that help users navigate more easily within a section or a page).

2. **Consistency**
   In most cases, it could be reached by using a formal set of guidelines for how to design products for a particular device or format.

3. **Confirmation**
   Require confirmation for any important or irreversible action to prevent errors user can accidentally make.

4. **User Control**
   'Undo', 'back', 'search' buttons, as well as keyboard shortcuts, are a great way to give a user the control over a website or an application.

5. **Accessibility**
   It's crucial for the product to easy-to-use by as many people as possible. UX design should remove the obstacles for people when they use the product, whether those obstacles are temporary or more permanent.

UX design balances business, people and technology. While it is true that a product cannot succeed without a healthy business, a business cannot succeed without a happy customer—and it is the UX Designer's job to make the customer happy.

UX designer Software Use?

1. Adobe XD
2. Axure
3. Figma

Interaction design

It is well recognized that component of interaction design is an essential part of user experience (UX) design, centering on the interaction between users and products [8]. The goal of interaction design is to create a product that produces an efficient and delightful end-user experience by enabling users to achieve their objectives in the best way possible[9] [10]

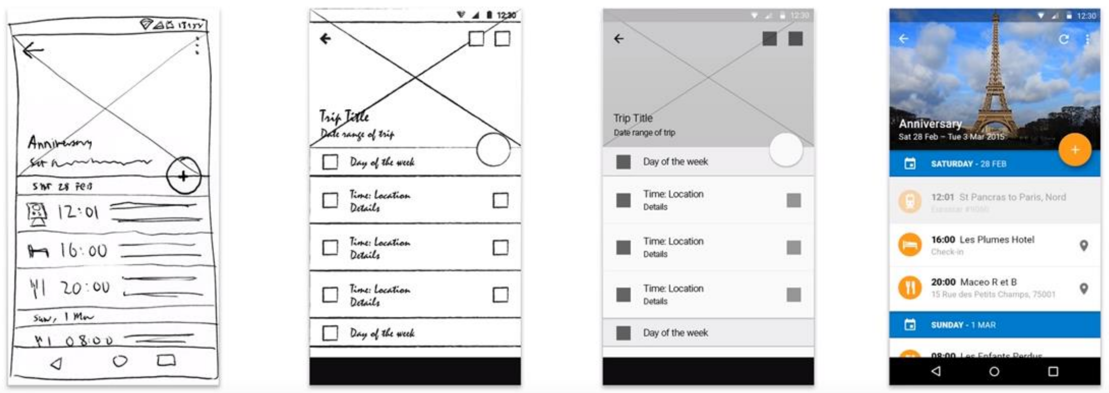
The current high emphasis on user-centered design and the strong focus on enhancing user experience have made interaction designers critical in conceptualizing products to match user expectations and meet the standards of the latest UI patterns and components.[11]To enable a pleasurable and desirable end user experience, the following are some considerations for the interaction design process:

- Defining interaction patterns best suited in the context
- Incorporating user needs collected during user research into the designs
- Features and information that are important to the user
- Interface behavior like drag-drop, selections, and mouse-over actions
- Effectively communicating strengths of the system
- Making the interface intuitive by building affordances
- Maintaining consistency throughout the system.
- Utilizing a haptic feedback system to reduce confusion[12]

In the last few years, the role of interaction designer has shifted from being just focused on specifying UI components and communicating them to the engineers to a situation now where designers have more freedom to design contextual interfaces which are based on helping meet the user needs.[13] Therefore, User Experience Design evolved into a multidisciplinary design branch that involves multiple technical aspects from motion graphics design and animation to programming.

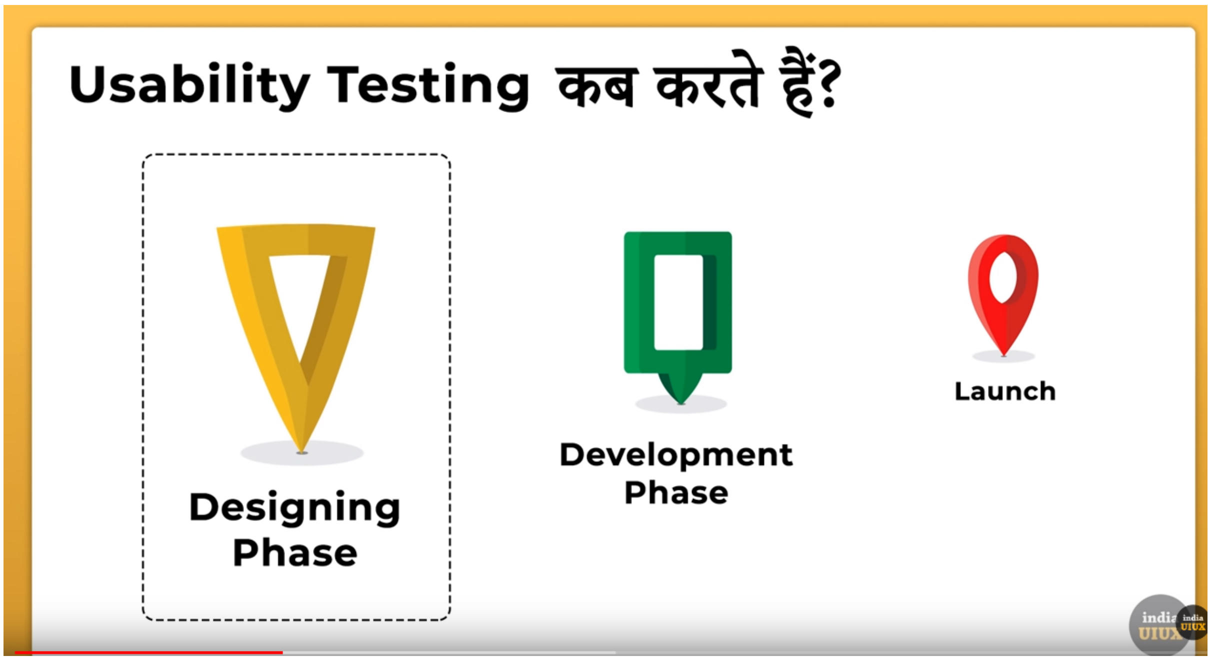
Wireframes कितने प्रकार के होते हैं?

Paper Wireframe | Lo-Fidelity Wireframe | Hi-Fidelity Wireframe | Final Mockup



USABILITY TESTING

- Meet User Expectations
- Easy & Simple
- Engaging


Usability Testing कब करते हैं?

1. What is a Persona?

- The Persona definition is that one or several fictional characters that can represent the majority of the potential users of product with conventional user demands and they are created through a great amount of quantitative and qualitative research. Persona answers the question “Who do we design for?” It is a powerful tool based on research findings in helping product function creation by optimizing the UX research and it not only represents a specific user but all of them, it can be understood as a typical character of the behavior, attitude, skills and contexts of all potential users.
- https://www.youtube.com/watch?v=vZ578SqL1oA

#Java Script

Question 44. How To Embed Javascript In A Web Page?
Answer :
javascript code can be embedded in a web page between
<script langugage="javascript"></script> tags

Question 47. What Boolean Operators Does Javascript Support?
Answer :

- &&
- ||
- !

Question 48. What Does "1"+2+4 Evaluate To?
Answer :
Since 1 is a string, everything is a string, so the result is 124.

Question 54. What Is The Difference Between An Alert Box And A Confirmation Box?
Answer :
An alert box displays only one button which is the OK button whereas the Confirm box displays two buttons namely OK and cancel.

Question 55. What Is A Prompt Box?
Answer :
A prompt box allows the user to enter input by providing a text box.

Question 56. Can Javascript Code Be Broken In Different Lines?
Answer :
Breaking is possible within a string statement by using a backslash at the end but not within any other javascript statement.that is ,
document.write("Hello world");
is possible but not document.write
("hello world");

Question 57. Taking A Developer's Perspective, Do You Think That That Javascript Is Easy To Learn And Use?
Answer :
One of the reasons JavaScript has the word "script" in it is that as a programming language, the vocabulary of the core language is compact compared to full-fledged programming languages. If you already program in Java or C, you actually have to unlearn some concepts that had been beaten into you. For example, JavaScript is a loosely typed language, which means that a variable doesn't care if it's holding a

string, a number, or a reference to an object; the same variable can even change what type of data it holds while a script runs.

The other part of JavaScript implementation in browsers that makes it easier to learn is that most of the objects you script are pre-defined for the author, and they largely represent physical things you can see on a page: a text box, an image, and so on. It's easier to say, "OK, these are the things I'm working with and I'll use scripting to make them do such and such," instead of having to dream up the user interface, conceive of and code objects, and handle the interaction between objects and users. With scripting, you tend to write a _lot_ less code.

What are Events?

All the different visitors' actions that a web page can respond to are called events.

An event represents the precise moment when something happens.

Examples:

- moving a mouse over an element
- selecting a radio button
- clicking on an element

The term "fires/fired" is often used with events. Example: "The keypress event is fired, the moment you press a key".

Here are some common DOM events:

Mouse Events	**Keyboard Events**	**Form Events**	**Document/Window Events**
click	keypress	submit	load
dblclick	keydown	change	resize
mouseenter	keyup	focus	scroll
mouseleave		blur	unload

CSS

What are Pseudo-classes?

A pseudo-class is used to define a special state of an element.

For example, it can be used to:

- Style an element when a user mouses over it
- Style visited and unvisited links differently

- Style an element when it gets focus

Syntax

The syntax of pseudo-classes:

```
selector:pseudo-class {
  property:value;
}
```

All CSS Pseudo Classes

Selector	Example	Example description
:active	a:active	Selects the active link
:checked	input:checked	Selects every checked <input> element
:disabled	input:disabled	Selects every disabled <input> element
:empty	p:empty	Selects every <p> element that has no children
:enabled	input:enabled	Selects every enabled <input> element
:first-child	p:first-child	Selects every <p> elements that is the first child of its parent
:first-of-type	p:first-of-type	Selects every <p> element that is the first <p> element of its parent
:focus	input:focus	Selects the <input> element that has focus
:hover	a:hover	Selects links on mouse over
:in-range	input:in-range	Selects <input> elements with a value within a specified range
:invalid	input:invalid	Selects all <input> elements with an invalid value
:lang(language)	p:lang(it)	Selects every <p> element with a lang attribute value starting with "it"
:last-child	p:last-child	Selects every <p> elements that is the last child of its parent
:last-of-type	p:last-of-type	Selects every <p> element that is the last <p> element of its parent
:link	a:link	Selects all unvisited links
:not(selector)	:not(p)	Selects every element that is not a <p> element
:nth-child(n)	p:nth-child(2)	Selects every <p> element that is the second child of its parent
:nth-last-child(n)	p:nth-last-child(2)	Selects every <p> element that is the second child of its parent, counting from the last child
:nth-last-of-type(n)	p:nth-last-of-type(2)	Selects every <p> element that is the second <p> element of its parent, counting from the last child
:nth-of-type(n)	p:nth-of-type(2)	Selects every <p> element that is the second <p> element of its parent
:only-of-type	p:only-of-type	Selects every <p> element that is the only <p> element of its parent
:only-child	p:only-child	Selects every <p> element that is the only child of its parent
:optional	input:optional	Selects <input> elements with no "required" attribute
:out-of-range	input:out-of-range	Selects <input> elements with a value outside a specified range
:read-only	input:read-only	Selects <input> elements with a "readonly" attribute specified
:read-write	input:read-write	Selects <input> elements with no "readonly" attribute
:required	input:required	Selects <input> elements with a "required" attribute specified

:root	root	Selects the document's root element
:target	#news:target	Selects the current active #news element (clicked on a URL containing that anchor name)
:valid	input:valid	Selects all <input> elements with a valid value
:visited	a:visited	Selects all visited links

What are Pseudo-Elements?

A CSS pseudo-element is used to style specified parts of an element.

For example, it can be used to:

- Style the first letter, or line, of an element
- Insert content before, or after, the content of an element

All CSS Pseudo Elements

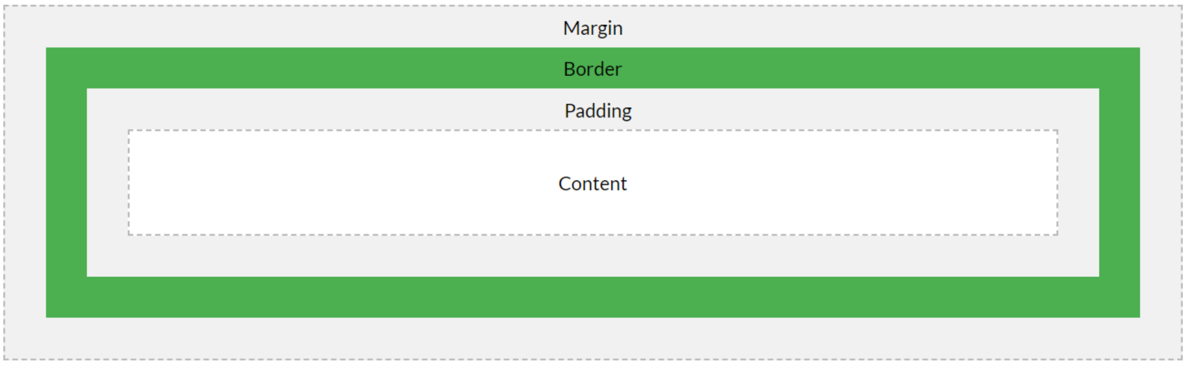
Selector	**Example**	**Example**	**description**
::after		p::after	Insert something after the content of each <p> element
::before		p::before	Insert something before the content of each <p> element
::first-letter		p::first-letter	Selects the first letter of each <p> element
::first-line		p::first-line	Selects the first line of each <p> element
::selection		p::selection	Selects the portion of an element that is selected by a user

link to more: https://www.w3schools.com/css/css_pseudo_classes.asp

The CSS Box Model

All HTML elements can be considered as boxes. In CSS, the term "box model" is used when talking about design and layout.

The CSS box model is essentially a box that wraps around every HTML element. It consists of: margins, borders, padding, and the actual content. The image below illustrates the box model:



What is Specificity?

If there are two or more conflicting CSS rules that point to the same element, the browser follows some rules to determine which one is most specific and therefore wins out.

Think of specificity as a score/rank that determines which style declarations are ultimately applied to an element.

The universal selector (*) has low specificity, while ID selectors are highly specific!

Note: Specificity is a common reason why your CSS-rules don't apply to some elements, although you think they should.

Specificity Hierarchy

Every selector has its place in the specificity hierarchy. There are four categories which define the specificity level of a selector:

Inline styles - An inline style is attached directly to the element to be styled. Example: <h1 style="color: #ffffff;">.

IDs - An ID is a unique identifier for the page elements, such as #navbar.

Classes, attributes and pseudo-classes - This category includes .classes, [attributes] and pseudo-classes such as :hover, :focus etc.

Elements and pseudo-elements - This category includes element names and pseudo-elements, such as h1, div, :before and :after.

How to Calculate Specificity?

Memorize how to calculate specificity!

Start at 0, add 1000 for style attribute, add 100 for each ID, add 10 for each attribute, class or pseudo-class, add 1 for each element name or pseudo-element.

Consider these three code fragments:

https://www.w3schools.com/css/css_specificity.asp

React JS

https://www.edureka.co/blog/interview-questions/react-interview-questions/

Explain Error Boundaries?

7. What do you understand by Virtual DOM? Explain its working.

A virtual DOM is a lightweight JavaScript object which originally is just the copy of the real DOM.

Java Script

What is “this” keyword in JavaScript

this keyword refers to an object, that object which is executing the current bit of javascript code.

In other words, every javascript function while executing has a reference to its current execution context, called ***this***. Execution context means here is how the function is called.

To understand *this* keyword, only we need to know how, when and from where the function is called, does not matter how and where function is declared or defined.

```
function bike() {
  console.log(this.name);
}

var name = "Ninja";
var obj1 = { name: "Pulsar", bike: bike };
var obj2 = { name: "Gixxer", bike: bike };

bike();          // "Ninja"
obj1.bike();     // "Pulsar"
obj2.bike();     // "Gixxer"
```

In the above code snippet, the job of `bike()` function is printing the `this.name` which means it’s trying to print the value of `name` property of the current execution context(i.e.*this* object).
In the above code snippet, when function `bike()` gets called it prints `“Ninja”`because the context of execution is not specified so by default its global context and there is a variable `name` is present at global context whose value is `“Ninja”`.
In case of `obj1().bike()` call, `“Pulsar”` gets printed and the reason behind this is function `bike()` gets called with the execution context as `obj1` so `this.name` became `obj1.name` . Same

with `obj2.bike()` call where the execution context of function `bike()` is `obj2`.

What is a Callback?

Simply put: *A callback is a function that is to be executed* ***after*** *another function has finished executing — hence the name 'call back'.*

More complexly put: *In JavaScript, functions are objects. Because of this, functions can take functions as arguments, and can be returned by other functions. Functions that do this are called* ***higher-order functions****. Any function that is passed as an argument is called a* ***callback function****.*

^ That's a lot of words. Lets look at some examples to break this down a little more.

Why use callbacks?

Callbacks are used in two different ways — in *synchronous* functions and *asynchronous* functions.

Callbacks in synchronous functions

If your code executes in a *top to bottom*, *left to right* fashion, *sequentially*, *waiting* until one code has *finished* before the next line begins, your code is **synchronous**.

Let’s look at an example to make it easier to understand:

```
const addOne = (n) => n + 1
addOne(1) // 2
addOne(2) // 3
addOne(3) // 4
addOne(4) // 5
```

In this example above, `addOne(1)` executes first. Once it’s done, `addOne(2)` begins to execute. Once `addOne(2)` is done, `addOne(3)` executes. This process goes on until the last line of code gets executed.

Callbacks are used in **synchronous** functions when you want a part of the code to be *easily swapped* with something else.

So, back in the `Array.filter` example above, although we filtered the array to contain numbers that are lesser than five, you could easily reuse `Array.filter` to obtain an array of numbers that are greater than ten:

What is undefined?

Undefined most typically means a variable has been declared, but not defined. For example:

```
let b;
console.log(b);
// undefined
```

You can also explicitly set a variable to equal undefined:

```
let c = undefined;
console.log(c);
// undefined
```

Finally, when looking up non-existent properties in an object, you will receive undefined:

```
var d = {};
console.log(d.fake);
// undefined
```

What is null?

There are two features of `null` you should understand:

- `null` is an empty or non-existent value.
- `null` must be assigned.

Here's an example. We assign the value of `null` to `a`:

```
let a = null;
console.log(a);
// null
```

Undefined:-

Undefined means a variable has been declared but has not yet been assigned a value.

Null:-

On the other hand, null is an assignment value. It can be assigned to a variable as a representation of no value.

JavaScript this keyword

The this keyword is a reference variable that refers to the current object. Here, we will learn about this keyword with help of different examples.

JavaScript this Keyword Example

Let's see a simple example of this keyword.

```
<script>
var address=
{
company:"Javatpoint",
city:"Noida",
state:"UP",
fullAddress:function()
{
return this.company+" "+this.city+" "+this.state;
}
};


var fetch=address.fullAddress();
document.writeln(fetch);

</script>
```

Output:

```
Javatpoint Noida UP
```

The following ways can be used to know which object is referred by this keyword.

https://www.w3schools.com/js/js_cookies.asp

What is **NaN** in JavaScript?

NaN is a short form of Not a Number. When a string or something else is being converted into a number and that cannot be done, then we get to see Nan.