	Python	JavaScript/TypeScript	Rust	Scala	Haskell
paradigms	OO, imperative, functional	OO, imperative, functional, generic	concurrent, functional, generic, imperative, structured, expression-based	functional/OO hybrid, expression-based	pure functional, expression-based
typing discipline	dynamic, duck, goose, gradual	JS: dynamic, structural, duck, gradual TS: dynamic, inferred, structural, duck, gradual	static, inferred, nominal, static, affine	static, inferred, nominal with structural features	static, inferred
install	pyenv install m.m.p pyenv rehash pyenv shell m.m.p python -m venv --prompt ${PWD##*/} .venv source .venv/bin/activate	curl -o- https://raw.githubusercontent.com/nvm-sh/nvm/v0.39.1/install.sh | bash nvm install 16 nvm use 16 TS: npm install -g typescript	curl --proto '=https' --tlsv1.2 https://sh.rustup.rs -sSf | sh rustup update	brew install sbt http://docs.scala-lang.org/getting-started.html	GHCup or brew install stack https://docs.haskellstack.org/en/stable/README/
docs	https://docs.python.org	https://developer.mozilla.org/en-US/docs/Web/JavaScript https://www.typescriptlang.org/docs/handbook/	https://www.rust-lang.org/learn https://doc.rust-lang.org/book/	https://www.scala-lang.org/api/current/	https://www.haskell.org/hoogle/
hyperpolyglot	http://hyperpolyglot.org/scripting	https://hyperpolyglot.org/gui https://hyperpolyglot.org/web	https://hyperpolyglot.org/rust	http://hyperpolyglot.org/rust	http://hyperpolyglot.org/ml
libraries	https://github.com/vinta/awesome-python https://modernpythonprojects.com/resources/	https://github.com/sorrycc/awesome-javascript https://github.com/dzharii/awesome-typescript	https://github.com/rust-unofficial/awesome-rust	https://github.com/lauris/awesome-scala	https://github.com/krispo/awesome-haskell
env	conda create --name snakes python=3.8 conda activate <project_name> conda install <package_name> conda env list pipenv				
build/env tooling	uv poetry	npm husky	cargo	sbt	stack
new project	cookiecutter hypermodern python	npm init tsc --init	cargo new my-project	sbt new scala/scala-seed.g8	stack new my-project cd my-project stack setup stack build stack exec my-project-exe
dep file	requirements.in requirements.txt requirements-dev.txt	package.json	Cargo.toml	build.sbt	
dep update	uv poetry pip-tools	npm i -g npm-upgrade npm-upgrade	cargo update cargo update <dep>		

interpreter	$ python foo.py	$ node foo.js
		
		$ ts-node foo.ts
			n/a	$ scala Foo.scala	runhaskell Main.hs

repl	$ python
	>>> exit()
	>>> <ctrl-d>
	
	ipython, bpython
		$ node
		>>> <ctrl-d>
		
		https://github.com/TypeStrong/ts-node
		$ ts-node
		>>> <ctrl-d>
			n/a	$ sbt console
				$ sbt consoleQuick
					$ stack ghci
					$ stack repl
					Prelude> ctrl-d

jupyter	default	IJavascript https://github.com/n-riesco/ijavascript tslab (unmaintained) https://github.com/yunabe/tslab	Evcxr https://github.com/google/evcxr	Almond https://almond.sh/	IHaskell https://github.com/gibiansky/IHaskell
online repl	https://www.python.org/shell/	https://repljs.com https://www.typescriptlang.org/play	https://play.rust-lang.org/	https://scastie.scala-lang.org/	
repl commands	jupyter x? — introspection x?? — source code %run file.py %load file.py %cpaste <paste> --	TS: > .type x	n/a	> :load foo.scala > :t foo _	-- load file Prelude> :l foo.hs -- reload file Foo> :r -- type Foo> :t map -- kind Foo> :k Num
multiline repl			n/a	> :paste	> :{ … :}
compile to executable	n/a	n/a	rustc main.rs cargo build cargo run	$ scalac Foo.scala $ scala -cp . Foo	$ ghc foo.hs $ ./foo
define module	module per .py file	module per .js/.ts file	module per .rs file	package foo.bar (in /foo/bar/_.scala)	module Foo.Bar where
use module	from collections import deque import collections from collections import *	import AB from './modules/ab.mjs' import { a as aa, b as bb } from './modules/ab.mjs' export const CD = 'Module CD'	use std::io;	import foo.Bar import foo.{Bar, Baz} import foo._	import Foo.Bar -- all functions in Bar now in scope without qualification
prelude	built-in functions	built-in functions	none	import java.lang._ import scala._ import scala.Predef._	Prelude
naming conventions	UpperCamelCase types snake_case values	UpperCamelCase types lowerCamelCase values	UpperCamelCase - types snake_case - variables UPPER_SNAKE_CASE - constants	UpperCamelCase type-level lowerCamelCase value-level	snake_case
evaluation	eager	eager	eager	eager pass-by-name Cats Eval	lazy
principles		Declare variables with let or const, not var. Use strict mode. Know your types and avoid automatic type conversion. Understand prototypes, but use modern syntax for classes, constructors, and methods. Don't use this outside constructors or methods.	* same-block variable shadowing * pattern matches and if have "arms" * ownership/borrowing * enums have variants * move vs. borrow	* implicits - abstract over context * for-comprehensions to process wrapper types	pure functional
Libraries					
testing	pytest				
files	pathlib.Path("./x.txt").read_text() pathlib.Path("./x.txt").write_text()			better-files	
CLI	click docopt argparse (ootb)		clap	scopt	
HTTP	requests httpx aiohttp	got	reqwest	akka-http http4s	
functional	returns pymonad pyrsistent	https://github.com/stoeffel/awesome-fp-js JS: Ramda TS: fp-ts, immutable, immer, ts-pattern		Cats scalaz	n/a
SQL	SQLAlchemy SQLModel asyncpg buildpg			doobie	esqueleto
Basic Syntax					

main	def main():
	println("a")
	
	if __name__ == '__main__':
	main()
		n/a	fn main() {
			
			}
				object Hello extends App {
				println("a")
				}
				
				object Hello {
				def main(args: Array[String]) {
				println("a")
				}
				}
					main = do
					putStrLn("a")

sequenced computation	default	default	default	default, but idiomatic with for comprehension:
				for {
				a <- f1()
				b <- f2(a)
				c <- f3(b)
				} yield {
				c
				}
					do
					expr1
					expr2

line comment	#	//	//	//	-- comment
block comment	n/a	/* */	/* */	/* */	{- comment another comment -}
line sep / statement terminator	; or newline newlines not separators inside (), [], {}, triple quote literals, or after backslash: \	; or newline newline not separator inside (), [], {}, "", '', or after binary operator newline sometimes not separator when following line would not parse as a valid statement	semicolon semicolon turns an expression into a statement!1	; or sometimes newline A newline does not terminate a statement: (1) inside ( ) or [ ], (2) if the preceding line is not a complete statement, (3) if following token not legal at start of a statement.	next line has equal or less indentation, or ;
scoped block	n/a	{ }	{ }	{ }	offside rule or { } f x = x + r where r = 42

deep copy	y = copy.copy(x)
	y = copy.deepcopy(x)
		structuredClone(x)	x.clone()	x.clone() // discouraged
				x.copy() // (case classes)	

Type system					
types	(optional, validated with mypy) in "typing" module None bytes int float bool str list dict set frozenset array int float complex fractions rationals decimal	(dynamic) undefined null string number boolean object symbol bigint TS adds: unknown never void	u8,u16,u32,u64,u128,usize i8,i16,i32,i64,i128,isize f32,f64 bool char decimals allow underscores str (literal compile-time stack) String (dynamic, heap)	Int Integer Long Float Double BigDecimal BigInt Fractional	Bool Int Int32 Word Integer Float Double Rational Fixed Ex Scientific
null	None	null		null scala.Null None	n/a
self type	typing.Self		Self		
intersection	n/a	TS: Point & { color: string }			
union	typing.Union[X, Y] X | Y	TS: string | number			
immutable variable	n/a ootb typing.Final annotation	const x = 1 TS: const x: number = 1	let x = ""; const FOO_BAR = ""; is an expression in if and while	val x = 1	x = 1 let x = 1
mutable variable	x = 1	let x = 1 TS: let x: number = 1	let mut x = "";	var x = 1	lol
assignment destructuring	x, y = xs	const x = 1, y = 2, z = 3 const o = { x, y, z }	let (x, y) = (1, 2);		
object literals	n/a	const o = { x: 1, y: 2, z: 3 } const {x, y, z} = o const {q = '10'} = o	let Foo { x, y, .. } = Foo { x: 1, y: 1, z:3 } let Foo { x: a, y: b } = Foo { x: 5, y: 1 } fn f(&(x, y): &(i32, i32)) {} fn f(Foo { x, y, .. }: Foo) {} let [x1, x2, ..] = [1, 2, 3, 4, 5]; let [a, middle @ .., z] = &xs		
named tuple	X = namedtuple(X, ['x', 'y']) p = X(11, y=22)				
lazy eval variable	n/a can be simulated with lambda	n/a		lazy val x = 1	
explicit type definition	x: int = 1	TS: const x: number = 1		1 : Int val x: Int = 1	1 :: Int
boolean	True False	true false		true false	data Bool = True | False
falsey	False, None, 0, 0.0, 0j, Decimal(0), Fraction (0,1), '', (), [], {}, set(), range(0)	false null undefined "" 0 NaN []		false	False
logical	not or and	! || &&	! || &&	! || &&	not || &&
bigint	int has unlimited precision	1n BigInt(1)			
symbol	n/a	Symbol("x")			
unknown	n/a	JS: n/a TS: unknown			
top/any	Any	TS: any		Any	
bottom	Any	TS: never also used when never returns		scala.Nothing	undefined
void/unit	None	TS: void	() - unit	() — Unit	
null test	x is None	x === null		x == null	n/a
arithmetic	+ - * / // %	+ - * / % ** ++ --		+ - * / %	+ - * / div mod quot rem
value comparators	== != < > <= >=	=== !== < > >= <= (excluding Objects) with type coercion: == !=		== != < > <= >=	== /= > < >= <=
object identity comparators	is is not	=== !==			
type determination	isinstance(x, int) isinstance(x, (int, str)) isinstance(x, Foo)	typeof x === 'string || x instanceof String			
type assertions	n/a	TS: x as string <string>x			
literal types	n/a	TS: let name: 'Foo Bar'; // type is 'Foo Bar'			
type guard	n/a	TS: "attr1" in x (o: any): o is Foo => !!o && "attr1"			
Error Handling					
end program			panic!("msg"); std::process::exit(0);		
result monad					
			r.unwrap_or_else(|x| panic!("msg")) r.unwrap() r.expect("msg") r?		
Control Structures					

if (statement or expression)	statement
	
	if p:
	e1
	elif p2:
	e2
	else:
	e3
		statement
		
		if (b) {
		} else if (b2) {
		else {
		}
			expression
			
			let z = if b {
			} else if b2 {
			} else {
			}
			
			let x = if b { 1 } else { 2 };
				expression
				
				if (p) then e1 else e2
				if (p) {
				e1
				} else {
				e2
				}
					expression
					
					if c then e1 else e2

ternary expression	True if True else False <expr1> if <conditional_expr> else <expr2>	x ? x : !x	if true { x } else { y };	if (true) x else y	
for	for i in range(1, 11): ...	for (let i = 1; i <= 10; i++) { }	for i in 1..=10 { ... }		

loop			loop {
			if x == 1 { break; }
			else continue;
			}
			
			let x = loop {
			break 1;
			}
			
			`loop1: loop {
			break 'loop1;
			}		

while	while cond: ...	while (cond) {
		}
			while cond {
			}
			
			while let x = expr {
			}
			
			while let Some(x) = expr {
			// do something with x
			}		

do/while		do {
		f()
		} while (cond)
			loop {
			f();
			if !condition {
			break;
			}
			}		

for/in/of container		for (const x of xs) for (const attr in obj) //attrs	for x in xs {} for (i, &x) in xs.iter().enumerate() {}		
guard blocks					f x | x > 0 = True | otherwise = False

switch		switch (x) {
		case "a":
		a = 0
		break
		case "b":
		a = 1
		break
		default:
		a = 2
		break
		}
		
		switch (x) {
		case "a": return 1;
		case "b": return 2;
		}			

pattern matching		TS w/ ts-pattern
		
		import { match, P } from 'ts-pattern';
		
		type Data =
		| { type: 'text'; content: string }
		| { type: 'img'; src: string };
		
		type Result =
		| { type: 'ok'; data: Data }
		| { type: 'error'; error: Error };
		
		const result: Result = ...;
		
		return match(result)
		.with({ type: 'error' }, () => `<p>Oops! An error occurred</p>`)
		.with({ type: 'ok', data: { type: 'text' } }, (res) => `<p>${res.data.content}</p>`)
		.with({ type: 'ok', data: { type: 'img', src: P.select() } }, (src) => `<img src=${src} />`)
		.exhaustive();
			match x {
			Some(50) => "50",
			Some(y) => y,
			0 => "zero",
			1 | 3 => "odd",
			4..=9 => "inclusive range",
			Point { x, y: 0 } => "destructure",
			
			_ => "hole!", // can't use _
			}
			
			if let Some(x) = Some(1) {
			println!("The maximum is configured to be {}", x);
			} else {
			}
				x match {
				case Nil => "empty list"
				case x::xs => "non-empty list"
				case 0 => "zero"
				case 1 | 3 => "odd"
				case x if x.foo > 42 "y"
				case b : Bar => b
				case foo @ Foo(_, bar @ Bar(v)) => (foo, v)
				case _ => "hole!" // can't use _
				}
					f t@(a, _) = do
					print a
					return t
					
					case x of
					True -> "a"
					False -> "b"

function parameter destructure	n/a			n/a, but can do case blocks
				
				val f: (List) = {
				case Nil => true
				case _ => false
				}
					f :: [Char] -> [Char]
					f [] = []
					f x:xs = xs
					f x = [x]

try/catch	try:
	f()
	except ValueError as e:
	handle_exception()
	else:
	no_exception_occured()
	finally:
	regardless_now_do()
		try {
		} catch (e) {
		
		} finally {
		
		}
				Try { x() }
				Try = Success(x) | Failure(e)
				
				try {
				Some(x())
				} catch {
				case ioe: IOException =>
				logger.error(ioe)
				None
				case fnf: FileNotFoundException =>
				logger.error(fnf)
				None
				}	

raise exception	raise Exception()	throw Error("")			
comprehension	[f(x) for x in xs if p] #listcomp (f(x) for x in xs if p) #genexp			for { x <- xs y <- ys if x + y == 0 } yield (x, y)	[a | a <- xs, a <= n]
generator function	yield x yield from xs				
Common Actions					
print a string	print("a") print("a", end="")	console.log("a")		print("a") println("a")	print "a" putStrLn "a" putStrLn . show $ "a"
math package	import math https://docs.python.org/3/library/math.html	Math https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Math		import scala.math._ https://www.scala-lang.org/api/current/scala/math/index.html	
JSON	import json json.dumps(x) pretty print — json.dumps(x, sort_keys=True, indent=4) json.loads('') https://docs.python.org/3/library/json.html	JSON.parse(s) JSON.stringify(o)		circe spray-json monocle	
base64	import base64 base64.b64decode(str)				
random	import random random.randint(0, 99) random.random() random.gauss(0, 1)			val r = scala.util.Random r.nextInt(excl) r.nextDouble r.nextFloat Random.nextString(10) Random.alphanumeric.take(10).mkString	

read file	with open(path, 'r') as f:
	lines = [x.rstrip() for x in f]
	or lines.readlines()
	
	file.read()
			fs::read_to_string("hello.txt")	val lines = Source.fromFile(filename).getLines.(toList|toArray|mkString)
				
				val bufferedSource = Source.fromFile("example.txt")
				for (line <- bufferedSource.getLines) {
				println(line.toUpperCase)
				}
				
				bufferedSource.close
				
				betterfiles	

write file	with open(path, 'w') as f: w.writelines(xs) w.write(s)				
read from stdin				scala.io.StdIn.readLine()	
union		TS: string | number string | null			
Types					
immutable array	array.array	TS: string[] Foo[]	[1, 2, 3] [i32; 5] [0; 5]	Seq Array Vector	

	Python	JavaScript/TypeScript	Rust	Scala	Haskell
mutable array / vector	array.array	new Int32Array(1024) Int8Array Uint8Array Uint8ClampedArray Int16Array Uint16Array Int32Array Uint32Array Float32Array Float64Array Float32Array.of(a, b) Float32Array.from(xs, f) xs.subarray(start, stop) ArrayBuffer DataView	vec![1, 2, 3]	Vector(1, 2, 3) x +: xs // front xs :+ x // end	
array check		Array.isArray(xs)			
linked list	list		Vec<T>	List HList (shapeless)	list
destructuring		let [f, s] = [1, 2, 3] let [f, s, ...r] = [1, 2, 3, 4] let [, s, ...r] = [1, 2, 3, 4]			

set	set() or frozenset(x)
	{ x }
	setcomp {x for x in xs}
	
	S ∩ Z: s & z
	S ∪ Z: s | z
	S \ Z: s - z
	S Δ Z: s ^ z
	e ∈ S: e in s
	S ⊆ Z: s <= z
	
	s.intersection(it, ...)
	s.union(it, ...)
	s.clear()
	s.discard(e)
	s.remove(e) #KeyError if not
		new Set()
		new Set(iterable)
		new WeakSet()
		set.size
		set.add(x)
		set.delete(x)
		set.has(x)
		set.clear()
		Array.from(xs)
		
		// intersect can be simulated via
		const intersection = new Set([...mySet1].filter((x) => mySet2.has(x)));
		
		// difference can be simulated via
		const difference = new Set([...mySet1].filter((x) => !mySet2.has(x)));
				Set(1, 2, 3)
				
				S ∩ Z: s & z
				S ∪ Z: s | z
				S \ Z: s &~ z or s -- z
				S Δ Z: n/a
				e ∈ S: s.contains(e)
				S ⊆ Z: s.subsetOf(z)
				
				diff: &~
				+ add element
				++ add another set
				- remove
				-- remove set	

map/dictionary	{'x': 1, 'y': 2}
	iteration over keys
	d.clear()
	k in d
	del d[k]
	d.get(k, [default])
	d[k] # error if not contains
	d.items()
	d.keys()
	d.values()
	d.pop(k, [default])
	d[k]
	d.update(m, [**kargs])
	d.setdefault(k, [default])
	
	dictcomp {x : v(x) for x in xs}
		new Map([["a", 1], ["b", 2]])
		new WeakMap()
		{} // object, not recommended
		
		m.set(k, v)
		
		m.delete(k)
		m.clear()
		m.has(k)
		m.get(k)
		
		for (const [k, v] of map) {}
		map.forEach((k, v) => {})
		
		m.keys()
		m.values()
		m.entries()
		
		m.size
			use std::collections::HashMap;
			
			let mut m = HashMap::new();
			m.insert(String::from("x"), 1);
			m.insert(String::from("y"), 2);
			m.entry(String::from("x")).or_insert(3);
			
			let k = String::from("x");
			m.get(&k).copied().unwrap_or(0);
			
			for (k, v) in &m { }
			
			let count = m.entry(k).or_insert(0);
			*count += 1;
				Map("a" -> 1, "b" -> 2)
				map("c") // NSEE
				map get "c" // None
				Seq(Some("a"-> 1), Some("b"-> 2), None).flatten.toMap
				(Seq("a" -> 1) ++ opt.map { x => "b" -> x }).toMap	

dates and times	from datetime import datetime, date, time dt = datetime(2021, 1, 20, 12, 30, 21) dt.strftime('%m/%d/%Y %H:%M') datetime.strptime('20091031', '%Y%m%d') datetime.timedelta(17, 7179) datetime.fromisoformat('2017-01-01T12:30:59.000000') datetime.isoformat() <=3.6 python-dateutil	new Date(iso8601Str) new Date(msFromEpoch) [UTC] Date.parse(dateStr) UTC(year, zeroBasedMonth, day, hours, minutes, seconds, milliseconds) invalid values are adjusted getUTCFullYear() getUTCMonth() [0-based] getUTCDate() getUTCHours() getUTCMinutes() getUTCSeconds() getUTCMilliseconds() getUTCDay() getTime() - ms since epoch toISOString() - ISO8601			
current datetime		new Date() [UTC] Date.now()			
		toLocaleString(locale, options), toLocaleDateString(locale, options), toLocaleTimeString(locale, options)			
enum		enum Color { R, G, B, } enum ABCs { A = 1, B, C, } ABCs[2] enum Foo { Up = "UP", Down = "DOWN", } const enum — irreversable		sealed trait Color case object R extends Color case object G extends Color case object B extends Color	
Functions / Methods					

function definition	# must return
	
	def f(x): return x
	
	def f(a, *bs, c_kw_only, **d)
	
	def f(a, *, b_kw_only)
	
	def tag(a, /, b)
		// must return
		
		function f(x) {
		 return x
		}
		
		TS:
		function f(x: number) : number {
		 return x
		}
			// last expression or may return
			
			fn f(a: i32) -> i32 {
			  a + 1
			}
			
			// trait parameter
			pub fn f(x: &impl Foo) -> impl Bar {
			      x.to_bar();
			}
			
			pub fn f<T: Foo>(x: &T, y: &T) { }
			
			pub fn f(x: &(impl Foo + Bar)) { }
			
			fn f<T, U>(t: &T, u: &U) -> T
			where
			      T: Foo + Bar,
			      U: Bar + Baz, { }
				// last expression or may return
				
				def f(x: Int):Int = x
				val f = (i: Int) => i + 1
				val f: Int => Int = (i) => i + 1
				val f = (i: Int) => { i + 1 }
				val f: Int => Int = (i) => { i + 1 }
				def f[A](a: A): String = a.toString
					// last expression
					
					f :: Int -> Int
					f x = x
					let f x = x

function invocation	f() f(1) f(1, 2, 3) f(p1=1, p2=2, p3=3)	f(1)	f(1)	f f() f(1) f(1, 2, 3) f(x=1)	f 1
variadic parameter definition	def first_and_last(*args, **kwargs) # args is an array # kwargs is a map			def f(xs: String*)	
variadic parameter invocation	f(*xs)			f(xs:_*)	
function placeholder	def f: pass	function f(x) { }		def f = ???	f = undefined
lambda / anonymous function	lambda x, y: x + y	() => 1 x => x + x (x, y) => (x + y) (x, y) => { return x + y } (x, y) => ({}) (x, y = 0, ...rst) => ({}) "arrow function" this bound to whatever it is outside, return value is last expression function (x) { return x }		x => x + 1 (x, y) => x + y _ + _	\x -> x + 1 \x y -> x + y \x y -> +
curryable function				def f(a: Int)(b: Int): Int	by default
curry a function				f_2args.curried(1)	f_2args 1 (*) 1
partial application	functools.partial(f, arg)			f_2args(1, _:Int)	n/a
function composition				f compose g g andThen f	f . g
function application				f.apply(1)	f $ 1
example function composition					putStrLn . show . take 3 . reverse $ "Hello"
infix in prefix				n/a	(*) 42 42
prefix in infix				n/a	42 `f` 42
right associative				starts with : e.g., 1 :: Nil	
infix/method				a + b a.+(b) a.foo(b) a foo b	
tail recursive				last action is tail call, with @tailrec for validation	default

non-strict arguments				f(a: => T){ // thunk
				  lazy val aEvaled = a
				}	

generator function		function* x() {
		  for (let i = 1; i < 10; i++)
		    yield 1
		}			

List					
type	list (mutable)	immutable library: List import { List } from 'immutable';		List (immutable) Seq (immutable) Array (mutable) Vector (immutable)	List
other	pyrsistent: PVector				
type	list[T]	JS: n/a TS: List<T>		List[T]	List[T]
empty ds	[]	List([])		Nil List() Seq()	[]
construction	[1, 2, 3] list(1, 2, 3)	List([1, 2, 3])		Nil List(1,2,3) 1 :: 2 :: 3 :: Nil Seq(1, 2, 3) Seq()	[1, 2, 3, 4]
ith	xs[i]	xs.get(i)		xs(i)	xs !! i
is empty?	not xs	xs.size === 0		xs == Nil xs.isEmpty xs.nonEmpty	null xs
spread	*[1, 2]				
join	','.join(xs)	xs.join(',')		xs.mkString(",")	
slice	s[a:b:c]	xs.slice(start, stop)			
first / car	xs[0]	xs[0] xs.at(0)		xs.head xs.headOption	head xs
rest / cdr	xs[1:]	xs.slice(1)		xs.tail	tail xs
last	xs[-1]	xs.at(-1)			
last as list	xs[-1:]	xs.slice(-1)		xs.last xs.lastOption	last xs
all but last	xs[:-1]	xs.slice(0, -1)		xs.init	init xs
length/size	len(xs)	xs.size		xs.length	length xs
push - add to end		xs.push(x, y, z)			
pop - remove from end		xs.pop(x)			
shift - remove from front		xs.shift(x)			
cons (add to head)		xs.unshift(x)		4 :: xs	42 : xs
add to tail		xs.push(x)		xs :+ x xs ::: List(x)	xs ++ [x]
add to middle				val (ys, zs) = xs splitAt n ys ::: new_element :: zs	let (ys,zs) = splitAt n xs in ys ++ [new_element] ++ zs
min/max	min(xs) max(xs)	Math.min(...xs) Math.max(...xs)		xs.min xs.max	minimum xs maximum xs
		xs.concat(xs2, xs3)			
Collection Operations					
take	xs[0:n] more_itertools.take			xs take n	take 1 xs
drop	xs[n:]			xs drop n	drop 1 xs
concat	xs + xs2	xs.concat(x, y, xs2, xs3)		xs ++ xs2 List.concat(xs, xs2) xs ::: xs2	xs ++ xs2 concat xs xs2
reverse list	list(reversed(xs))			xs.reverse	reverse xs
xs w/ x in position n	xs[n] = x			xs updated (n, x)	let (ys,zs) = splitAt n xs in ys ++ [new_element] ++ (tail xs)
operations					
tuple	(42, 'foo', True)	TS: let x: [string, number] = ["hello", 10]	let t = ("a", 1) t == (t.0, t.1)	(42, "foo", true) nested pattern matching!	(1, "foo", True) (,) 1 "foo" True , is an infix operator!
tuple operators	t[0] # unpacking x, y, *rest = (1, 2, 3, 4, 5)	TS: x[0]		t._1 t._2	fst t snd t swap t
map with default value	collections.defaultdict(list)			Map().withDefaultValue("foo")	
add to mutable map	d['a'] = 1			m + ("a" -> 1) m + ("a", 1) m ++ Map("a" -> 1)	
deque	from collections import deque deque(range(10), maxlen=10) rotate, appendLeft, extend, extendLeft				

range	#start to at most stop by step
	range(start, stop, step)
	
	# generator
	itertools.count(start, step)
			(1..10) // 1, 2, ..., 9
			(1..=10) // 1, 2, ..., 10
				1 until 10
				1 to 10 by 2
				List.range(start, endExclusive)	

tabulate				List.tabulate(5)(n => n * n)	
collect				collect(pf) collectFirst(pf)	
lazy collections				xs.view...to(Seq) (2.13) LazyList Stream (older)	default
		for (const x of xs) { }			

		for (const i in xs) {
		  xs[i]
		}			

	https://docs.python.org/3/library/functools.html https://docs.python.org/3/library/itertools.html				

functions	operator
	      Math operations: add(), sub(), mul(), floordiv(), abs(), ...
	
	      Logical operations: not_(), truth().
	
	      Bitwise operations: and_(), or_(), invert().
	
	      Comparisons: eq(), ne(), lt(), le(), gt(), and ge().
	
	      Object identity: is_(), is_not().				

exists	any(xs)	xs.some(f)		xs exists p	
forall	all(xs)	xs.every(f)		xs forall p	all xs p
contains	x in xs	xs.includes(x)		xs contains x	any xs p elem x xs
index of		xs.indexOf(v) xs.findIndex(f)		xs.indexOf(x) xs.indexWhere(f)	
find		xs.find(f)		xs.find(f)	
lastIndexOf		xs.lastIndexOf(v)sta			
take while	itertools.takewhile(predicate, iter)			xs takeWhile p	
drop while	itertools.dropwhile(predicate, iter)			xs dropWhile p	
foreach		xs.forEach(f)		xs.foreach(f)	
span				xs span p	
sort with				xs sortWith p	
sort	sorted(xs) xs.sort (mutation)			xs.sorted	
group by	itertools.groupby(iter, key_func=None)			xs groupBy f	
map	map(f, iter1, iter2, ...)	xs.map(f)		xs map f	
flatmap		xs.flatMap(f)		xs flatMap f	
reduce	functools.reduce(function, iterable[, initializer]) # also foldl			reduce[B >: A](op: (B, B) => B): B xs reduce f xs reduceLeft f xs reduceRight f xs reduce*Option f	
fold left		xs.reduce((acc, curr) => acc + curr, init)		xs.foldLeft(init)(f) def foldLeft[B](z: B)(op: (B, A) => B): B	foldl f xs
fold right		xs.reduce((acc, curr) => acc + curr, init)		xs.foldRight(init)(f) def foldRight[B](z: B)(op: (A, B) => B): B	foldr f xs
fold (arbitrary)				xs.fold(init)(f) def fold[A1 >: A](z: A1)(op: (A1, A1) => A1): A1	
scan				scanRight scanLeft	
filter	filter(p, xs)	xs.filter(i => i === "a")		xs filter p	
filter not	itertools.filterfalse(predicate, iter)			xs filterNot p	
partition				xs partition p	
flatten	[i for sl in l for i in sl]	xs.flat()		xss.flatten	
zip	zip(iterA, iterB, ...)			xs zip ys	
unzip				xs.unzip	
sum	sum(iter, start=0)			xs.sum	sum xs
product				xs.product	product xs
	sorted(iterable, key=None, reverse=False)				
index and element	enumerate(iter, start=0)	for (const [i, x] of xs.entries())			
chain	itertools.chain(iterA, iterB, ...)				
combinations	itertools.combinations(iterable, r)				
permutations	itertools.permutations(iterable, r=None)				
	iter(x, stop)				
accumulate	itertools.accumulate(iterable[, func, *, initial=None])				
		every filter find findIndex findLast findLastIndex flatMap forEach group groupToMap map some reduce() reduceRight()			
splice		xs.splice(4, 5, x)			
reverse		xs.reverse()			
sort		xs.sort(f) // in-place			
Strings					
char	n/a	n/a	'a'	'a'	'a'

at		s.charAt(1) // only positive index
		s.at(-1) // pos or neg index
			s.chars().nth(0)		

literal string	'foo'
	"foo"
	'''foo'''
	"""foo"""
		'foo'
		"foo"
			"foo" // immutable str
			String::new() // mutable String
			String::from("foo")
			let s = "foo".to_string();
				"foo"
				"""multiline foo"""
					"foo"

interpolation	f'{x} {y}'	`${x} ${y}`	n/a	s"$foo ${bar.f} ${floatVal}%.2f" s""""in quotes"""" f"format string" TBD	

raw	r'a literal backslash n \n'	String.raw`x`	let x = r"\n or \t";
			let x = r#"x"y"x"#; // also multiline
			let complex = r##"xC:\Users\Name\Documents"##;
				raw"a literal backslash n \n"	

string conc.	"a" + "b"
	f"{a}{b}"
		"a" + "b"
		`${a}${b}`
			String::from("foo").push_str("bar")
			
			let s1 = String::from("foo");
			let s2 = String::from("bar");
			let s3 = s1 + &s2; // note: s1 out!
			
			let s = format!("{s1}-{s2}-{s3}");
				"a" + "b"	"a" ++ "b"

bytes	s.encode('utf-8') bs.decode('utf-8')		s.bytes()		
format string	'{0:.2f} {1:s} ${2:d}'.format(3.14159, 'x', 1) 'a: %d, b: %d' % (a, b) 'a: %(a)d, b: %(b)d' % {"a":a, "b":b}		format!("{}", x); format!("{x}", x=1); format!("{0} {1} {0}", 7, 8); {x} - std::fmt::Display {x:?} - Debug {:#?} - pretty Debug	"foo %s %d %.2f".format("bar", 7, 3.1415)	
casing	s.upper() s.lower()	s.toUpperCase() s.toLowerCase()		"foo".toUpperCase "FOO".toLowerCase	
trim	s.strip() s.lstrip() s.rstrip()	s.trim() s.trimStart() s.trimEnd()		" foo ".trim	
type conversion	int(s)			x.toString toInt toFloat	
split	s.split(',')	s.split(',') s.split('/', 1) // array len 1 result!	split(sep) split_inclusive(sep) rsplit(sep) split_terminator(sep) rsplit_terminator(sep)	"a b c".split(" ")	
partition	'a b c'.partition(' ') 'a b c'.rpartition(' ')				
length	len(s)	s.length		"foo".length	
substring	s[start:end]	s.substring(start, end)		"foo".substring(0, 1)	
slice	xs[-2:-1]	s.slice(-6, -2)			

<table>
<tr><th></th><th>Python</th><th>JavaScript/TypeScript</th><th>Rust</th><th>Scala</th><th>Haskell</th></tr>
<tr><td>regex</td><td>import re

re.search("foo.*", text)

m = re.match(pattern, string)
m.group(0)

r = re.compile(r'\w+')

r.search(s) - first anywhere in string
r.match(s) - only starting at beginning
r.fullmatch(s) - only matching entire string

r.split(s, maxsplit=1)
r.sub(repl, s, count=1)
re.escape(pattern)
groupdict
r.findall(s) / finditer

https://docs.python.org/3/library/re.html</td><td>const regex = /[A-Z]/g;
let r = /(.*/igm

r.test(s)
r.exec(s).groups

s.search(regex)
s.replace(regex, repl)
s.replaceAll(regex, repl)
s.match(regex)
s.matchAll(regex)</td><td></td><td>raw"[0-9]+".r.findFirstIn(s)
findAllIn
findFirstMatchIn
replaceAllIn

"2004-01-20" match {
  case raw"(\d{4})-(\d{2})-(\d{2})".r(year, month, day) => s"$year"
}

https://www.scala-lang.org/api/current/scala/util/matching/Regex.html</td><td></td></tr>
<tr><td>startswith</td><td>str.startswith(prefix[, start[, end]])</td><td>startsWith</td><td></td><td></td><td></td></tr>
<tr><td>endsWith</td><td>str.endswith(suffix[, start[, end]])</td><td>endsWith</td><td></td><td></td><td></td></tr>
<tr><td>find</td><td>str.find(sub[, start[, end]])</td><td>find</td><td></td><td></td><td></td></tr>
<tr><td>contains</td><td>x in s</td><td>s.includes(x)</td><td></td><td></td><td></td></tr>
<tr><td>repeat</td><td></td><td>'x'.repeat(3)</td><td></td><td></td><td></td></tr>
<tr><td>other</td><td></td><td>'x'.padStart(3, '>>>')
s.toUpperCase()
s.toLowerCase()</td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td>indexOf
lastIndexOf
s.match(regexp)
s.matchAll(regexp)</td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td>localeCompare</td><td></td><td></td><td></td></tr>
<tr><td>slice reference</td><td></td><td></td><td>let s = String::from("xyzzyxyzzy");
&s[0..5]
&s[0..]
&s[..5]
&s[..]</td><td></td><td></td></tr>
<tr><td>Monadic</td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td>option/maybe</td><td>from returns.maybe import Maybe, Some, Nothing, maybe

a1: Maybe[int] = Maybe.from_optional(1)
b1: Maybe[int] = Maybe.from_optional(None)

a2: Maybe[int] = Maybe.from_value(1)
b2: Maybe[int] = Maybe.from_value(None)</td><td></td><td>Option<T> = Some<T> | None</td><td>Option = None | Some(x)
getOrElse("<not assigned>")
map
match { case None ... case Some(x) }</td><td>data Maybe = Nothing | Just(x)</td></tr>
<tr><td>option example</td><td></td><td></td><td></td><td>def toInt(s: String): Option[Int] = {
  try {
    Some(s.toInt)
  } catch {
    case e: Exception => None
  }
}

toInt(aString) match {
  case Some(i) => println(i)
  case None => println("Error: Could not convert String to Int.")
}</td><td></td></tr>
<tr><td>either / result</td><td>from returns.result import Result, Success, Failure
a: Result[int, str] = Success(1)
b: Result[int, str] = Failure("error")

a.map(lambda n: n + 1) # Success(2)
b.map(lambda n: n + 1) # Failure("error")</td><td></td><td>Result<T,E> = Ok<T> | Err<E></td><td>Either = Left(a) | Right(b)
right-biased</td><td>data Either a b = Left a | Right b</td></tr>
<tr><td>try
(either w/ right bound to Exception type)</td><td></td><td></td><td></td><td>Try = Success(v) | Failure(e)</td><td></td></tr>
<tr><td>future</td><td></td><td></td><td></td><td>Future.successful(v)
Future.failed(v)</td><td></td></tr>
<tr><td>User-defined Types</td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td>type alias</td><td></td><td>TS:
type Name = string;</td><td></td><td>type Name = String</td><td>type Name = String</td></tr>
<tr><td>ADT / named tuple / struct</td><td>collections.namedtuple
typing.NamedTuple
@dataclasses.dataclass
attrs
pydantic</td><td>TS:
type Foo = {
  attr1: string;
  optAttr1?: string;
}

let x: Foo = { attr1: "x" }</td><td>struct Foo {
  x: u32,
  y: str,
}

pub struct Foo<T, U> {
    x: T,
    y: U,
}

impl<T, U> Foo<T, U> {
  pub fn new(x: T, y: U) -> Foo {
    Foo { x: x, y: y }
    // or Foo { x, y } for same name
  }

  pub fn value_x(&self) -> &T {
    &self.x
  }
}

impl Foo<i32, i32> {
  pub fn new(x: i32, y: i32) -> Foo {
    Foo { x, y }
  }

  pub fn value_x(&self) -> &i32 {
    &self.x
  }
}</td><td>case class Foo(attr1: String)</td><td></td></tr>
<tr><td>trait/interface</td><td></td><td>TS:
interface Foo {
  attr1: string;
}</td><td>pub trait Foo {
    fn f(&self) -> String;
}</td><td>trait Foo { }</td><td></td></tr>
<tr><td>class</td><td>class C2(C1) {
#todo
}</td><td>function Foo(attr1, attr2) {
  this.attr1 = attr1
  this.attr2 = attr2
}

Foo.prototype.getName = function() {
  return this.attr1 + ' ' + this.attr2
}

const foo = new Foo('a')

class Foo extends Bar implements Baz, Qux {
  attr1: string

  constructor(public readonly attr2: string){}

  f(p) {
    this.attr1 = p
  }

  static g(a) { return 1 }

  get attr1() { return attr1 }

  set attr1(x) { attr1 = x }
}

const foo = new Foo('a')

foo.attr1
foo.attr1 = "x"</td><td>n/a</td><td></td><td></td></tr>
<tr><td>attribute visibility modifiers</td><td></td><td>public
protected
private
readonly</td><td></td><td>(none – public)
protected
protected[package]
private</td><td></td></tr>
<tr><td>abstract class</td><td>from abc import ABC

class Foo(ABC):
  pass</td><td>abstract class Foo {}</td><td>n/a</td><td>abstract class Foo() { }</td><td></td></tr>
<tr><td>ADT product type</td><td>from collections import namedtuple
Foo = namedtuple('Foo', 'x y')

or

import typing
Foo = typing.NamedTuple('Foo', [('x', str), ('y', float)])
or
Foo = typing.NamedTuple('Foo', x=str, y=float)

or

from dataclasses import dataclass

@dataclass(frozen=True)</td><td></td><td></td><td>sealed trait Foo
final case class Bar extends Foo
final case class Baz(c: Int, d: String) extends Foo</td><td>data Foo = Foo String Int
data Foo = Foo { a :: String, b :: Int }</td></tr>
<tr><td>ADT sum (tagged union)</td><td></td><td></td><td></td><td>sealed trait Foo
case object Bar extends Foo
case object Baz extends Foo</td><td>data Bool = False | True</td></tr>
<tr><td>ADT product (record)</td><td></td><td></td><td></td><td></td><td>data Foo = Bar | Baz Corge deriving Grault</td></tr>
<tr><td>traits/mixins</td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td>single unary data constructor</td><td></td><td></td><td></td><td>@newtype case class C(v: String)</td><td>newtype</td></tr>
<tr><td>nullary data constructor</td><td></td><td></td><td></td><td>case object Foo</td><td>data Foo = Foo</td></tr>
<tr><td>unary data constructor</td><td></td><td></td><td></td><td>case class Foo(attr1: String)</td><td>data Foo a = Foo a</td></tr>
<tr><td>type parameters</td><td></td><td>TS:
interface Foo<T> {
    f(value: T): T;
}

function f<T> (x: T): T {
    return x;
}

const printMe = <T> (x: T): T => {
   return x;
}</td><td></td><td></td><td></td></tr>
<tr><td>arbitrary indexed properties</td><td></td><td>TS:
type Foo = {
 [arg: string]: string | string[]
}</td><td></td><td></td><td></td></tr>
<tr><td>Other</td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td>Blanket impl
impl<T: Display> ToString for T {
    // --snip--
}</td><td></td><td></td></tr>
<tr><td>derive</td><td></td><td></td><td>#[derive(Debug)]</td><td></td><td>deriving (Show, Eq, Ord)</td></tr>
<tr><td>debug logging</td><td></td><td></td><td>dbg!(&x);</td><td></td><td></td></tr>
<tr><td>Metaprogramming</td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td>Object behavior</td><td>https://docs.python.org/3/reference/datamodel.html</td><td>valueOf()

all in Symbol class
toStringTag
toPrimitive
species
iterator
asyncIterator
hasInstance
match, matchAll, replace, search, split
isConcatSpreadable</td><td></td><td></td><td></td></tr>
<tr><td>Tests</td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td>pytest

assert p

with pytest.raises(ZeroDivisionError):
        1 / 0

./tests/test_x.py

pytest tests/test_x.py::Class::method
pytest tests/test_x.py::function

pytest -k "MyClass and not method"

@pytest.mark.slow

pytest -m "mark1 and not mark2"

pytest --lf (last failed)

@pytest.mark.skip(reason)
@pytest.mark.skipif(reason)
@pytest.mark.xfail

--doctest-modules</td><td></td><td>cargo test

#[cfg(test)]
mod tests {
      use super::*;

      #[test]
      #[should_panic]
      fn f() {

      }
}

assert!(b, "msg {}", arg1)
assert_eq!(l, r)
assert_ne!(l, r)

#[should_panic(expected = "msg")]

fn t() -> Result<(), String> { }

cargo test -- --show-output
cargo test test1

#[ignore]</td><td></td><td></td></tr>
<tr><td>Array/Vector</td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td>class/struct</td><td>pyrsistent: PVector
array module</td><td>array (mutable)</td><td>native array
Vec</td><td>Vector (immutable)
Array (mutable)</td><td></td></tr>
<tr><td>type</td><td></td><td>JS: n/a
TS: T[]</td><td>Vec<T></td><td>List[T]</td><td></td></tr>
<tr><td>empty ds</td><td></td><td>[]</td><td>let a: [i32; 0] = [];

let xs: Vec<i32> = Vec::new();
let xs: Vec<i32> = vec![];</td><td>Nil
List()
Seq()</td><td></td></tr>
<tr><td>construction</td><td></td><td>[1, 2, 3]
Array.from(otherIterable)</td><td>[1, 2, 3, 4, 5]

vec![1, 2, 3]</td><td>Nil
List(1,2,3)
1 :: 2 :: 3 :: Nil
Seq(1, 2, 3)
Seq()</td><td></td></tr>
<tr><td>length/size</td><td></td><td>xs.length</td><td>xs.len()</td><td>xs.length</td><td></td></tr>
<tr><td>ith</td><td></td><td>xs[i]</td><td>let x: &i32 = &xs[2];
let maybe_x: Option<&i32> = xs.get(2);</td><td>xs(i)</td><td></td></tr>
<tr><td>iteration</td><td></td><td></td><td>let xs = vec![1, 2, 3];
for x in &xs { x; }

let mut xs = vec![1, 2, 3];
for x in &mut xs {
*x += 1;
}</td><td></td><td></td></tr>
<tr><td>spread</td><td></td><td>...[1, 2]</td><td></td><td></td><td></td></tr>
<tr><td>is empty?</td><td></td><td>!xs
xs.length === 0</td><td>xs.is_empty()</td><td>xs == Nil
xs.isEmpty
xs.nonEmpty</td><td></td></tr>
<tr><td>join</td><td></td><td>xs.join(',')</td><td></td><td>xs.mkString(",")</td><td></td></tr>
<tr><td>slice</td><td></td><td>xs.slice(start, stop)</td><td>&xs[a..b]</td><td></td><td></td></tr>
<tr><td>first / car</td><td></td><td>xs[0]
xs.at(0)</td><td></td><td>xs.head
xs.headOption</td><td></td></tr>
<tr><td>rest / cdr</td><td></td><td>xs.slice(1)</td><td></td><td>xs.tail</td><td></td></tr>
<tr><td>last</td><td></td><td>xs.at(-1)</td><td></td><td></td><td></td></tr>
<tr><td>last as list</td><td></td><td>xs.slice(-1)</td><td></td><td>xs.last
xs.lastOption</td><td></td></tr>
<tr><td>all but last</td><td></td><td>xs.slice(0, -1)</td><td></td><td>xs.init</td><td></td></tr>
<tr><td>push - add to end</td><td></td><td>xs.push(x)</td><td>let mut xs: Vec<i32> = vec![1, 2, 3];
xs.push(1);</td><td></td><td></td></tr>
<tr><td>pop - remove from end</td><td></td><td>xs.pop()</td><td>xs.pop();</td><td></td><td></td></tr>
<tr><td>shift - remove from front</td><td></td><td>xs.shift()</td><td></td><td></td><td></td></tr>
<tr><td>unshift - add to front</td><td></td><td>xs.unshift(x)</td><td></td><td></td><td></td></tr>
<tr><td>min/max</td><td></td><td>Math.min(...xs)
Math.max(...xs)</td><td></td><td>xs.min
xs.max</td><td></td></tr>
<tr><td>take</td><td></td><td></td><td></td><td>xs take n</td><td></td></tr>
<tr><td>drop</td><td></td><td></td><td></td><td>xs drop n</td><td></td></tr>
<tr><td>concat</td><td></td><td>xs.concat(x, y, xs2, xs3)</td><td></td><td>xs ++ xs2
List.concat(xs, xs2)
xs ::: xs2</td><td></td></tr>
<tr><td>reverse list</td><td></td><td></td><td></td><td>xs.reverse</td><td></td></tr>
<tr><td>xs w/ x in position n</td><td></td><td></td><td></td><td>xs updated (n, x)</td><td></td></tr>
<tr><td>cons (add to head)</td><td></td><td>xs.unshift(x)</td><td></td><td>4 :: xs</td><td></td></tr>
<tr><td>add to tail</td><td></td><td>xs.push(x)</td><td></td><td>xs :+ x
xs ::: List(x)</td><td></td></tr>
<tr><td>add to middle</td><td></td><td></td><td></td><td>val (ys, zs) = xs splitAt n
ys ::: new_element :: zs</td><td></td></tr>
<tr><td>operations</td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td>tuple</td><td></td><td>TS:
let x: [string, number] = ["hello", 10]</td><td></td><td>(42, "foo", true)
nested pattern matching!</td><td></td></tr>
<tr><td>tuple operators</td><td></td><td>TS:
x[0]</td><td></td><td>t._1
t._2</td><td></td></tr>
<tr><td>map with default value</td><td></td><td></td><td></td><td>Map().withDefaultValue("foo")</td><td></td></tr>
<tr><td>add to mutable map</td><td></td><td></td><td></td><td>m + ("a" -> 1)
m + ("a", 1)
m ++ Map("a" -> 1)</td><td></td></tr>
<tr><td>deque</td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td>range</td><td></td><td></td><td></td><td>1 until 10
1 to 10 by 2
List.range(start, endExclusive)</td><td></td></tr>
<tr><td>tabulate</td><td></td><td></td><td></td><td>List.tabulate(5)(n => n * n)</td><td></td></tr>
<tr><td>collect</td><td></td><td></td><td></td><td>collect(pf)
collectFirst(pf)</td><td></td></tr>
<tr><td>lazy collections</td><td></td><td></td><td></td><td>xs.view...to(Seq) (2.13)
LazyList
Stream (older)</td><td></td></tr>
<tr><td></td><td></td><td>for (const x of xs) {
}</td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td>for (const i in xs) {
  xs[i]
}</td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td>functions</td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td>exists</td><td></td><td>xs.some(f)</td><td></td><td>xs exists p</td><td></td></tr>
<tr><td>forall</td><td></td><td>xs.every(f)</td><td></td><td>xs forall p</td><td></td></tr>
<tr><td>contains</td><td></td><td>xs.includes(x)</td><td></td><td>xs contains x</td><td></td></tr>
<tr><td>index of</td><td></td><td>xs.indexOf(v)
xs.findIndex(f)</td><td></td><td>xs.indexOf(x)
xs.indexWhere(f)</td><td></td></tr>
<tr><td>find</td><td></td><td>xs.find(f)</td><td></td><td>xs.find(f)</td><td></td></tr>
<tr><td>lastIndexOf</td><td></td><td>xs.lastIndexOf(v)sta</td><td></td><td></td><td></td></tr>
<tr><td>take while</td><td></td><td></td><td></td><td>xs takeWhile p</td><td></td></tr>
<tr><td>drop while</td><td></td><td></td><td></td><td>xs dropWhile p</td><td></td></tr>
<tr><td>foreach</td><td></td><td>xs.forEach(f)</td><td></td><td>xs.foreach(f)</td><td></td></tr>
<tr><td>span</td><td></td><td></td><td></td><td>xs span p</td><td></td></tr>
<tr><td>sort with</td><td></td><td></td><td></td><td>xs sortWith p</td><td></td></tr>
<tr><td>sort</td><td></td><td></td><td></td><td>xs.sorted</td><td></td></tr>
<tr><td>group by</td><td></td><td></td><td></td><td>xs groupBy f</td><td></td></tr>
<tr><td>map</td><td></td><td>xs.map(f)</td><td></td><td>xs map f</td><td></td></tr>
<tr><td>flatmap</td><td></td><td>xs.flatMap(f)</td><td></td><td>xs flatMap f</td><td></td></tr>
<tr><td>reduce</td><td></td><td></td><td></td><td>reduce[B >: A](op: (B, B) => B): B
xs reduce f
xs reduceLeft f
xs reduceRight f
xs reduce*Option f</td><td></td></tr>
<tr><td>fold left</td><td></td><td>xs.reduce((acc, curr) => acc + curr, init)</td><td></td><td>xs.foldLeft(init)(f)
def foldLeft[B](z: B)(op: (B, A) => B): B</td><td></td></tr>
<tr><td>fold right</td><td></td><td>xs.reduce((acc, curr) => acc + curr, init)</td><td></td><td>xs.foldRight(init)(f)

def foldRight[B](z: B)(op: (A, B) => B): B</td><td></td></tr>
<tr><td>fold (arbitrary)</td><td></td><td></td><td></td><td>xs.fold(init)(f)
def fold[A1 >: A](z: A1)(op: (A1, A1) => A1): A1</td><td></td></tr>
<tr><td>scan</td><td></td><td></td><td></td><td>scanRight
scanLeft</td><td></td></tr>
<tr><td>filter</td><td></td><td>xs.filter(i => i === "a")</td><td></td><td>xs filter p</td><td></td></tr>
<tr><td>filter not</td><td></td><td></td><td></td><td>xs filterNot p</td><td></td></tr>
<tr><td>partition</td><td></td><td></td><td></td><td>xs partition p</td><td></td></tr>
<tr><td>flatten</td><td>def flatten(xs: list[list[Any]]) -> list[Any]:
    return [i for sl in xs for i in sl]</td><td>xs.flat()</td><td></td><td>xss.flatten</td><td></td></tr>
<tr><td>zip</td><td></td><td></td><td></td><td>xs zip ys</td><td></td></tr>
<tr><td>unzip</td><td></td><td></td><td></td><td>xs.unzip</td><td></td></tr>
<tr><td>sum</td><td></td><td></td><td></td><td>xs.sum</td><td></td></tr>
<tr><td>product</td><td></td><td></td><td></td><td>xs.product</td><td></td></tr>
<tr><td>index and element</td><td></td><td>for (const [i, x] of xs.entries())</td><td></td><td></td><td></td></tr>
<tr><td>chain</td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td>combinations</td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td>permutations</td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td>accumulate</td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td>every
filter
find
findIndex
findLast
findLastIndex
flatMap
forEach
group
groupToMap
map
some

reduce()

reduceRight()</td><td></td><td></td><td></td></tr>
<tr><td>splice</td><td></td><td>xs.splice(4, 5, x)</td><td></td><td></td><td></td></tr>
<tr><td>reverse</td><td></td><td>xs.reverse()</td><td></td><td></td><td></td></tr>
<tr><td>sort</td><td></td><td>xs.sort(f) // in-place</td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td></td></tr>
</table>