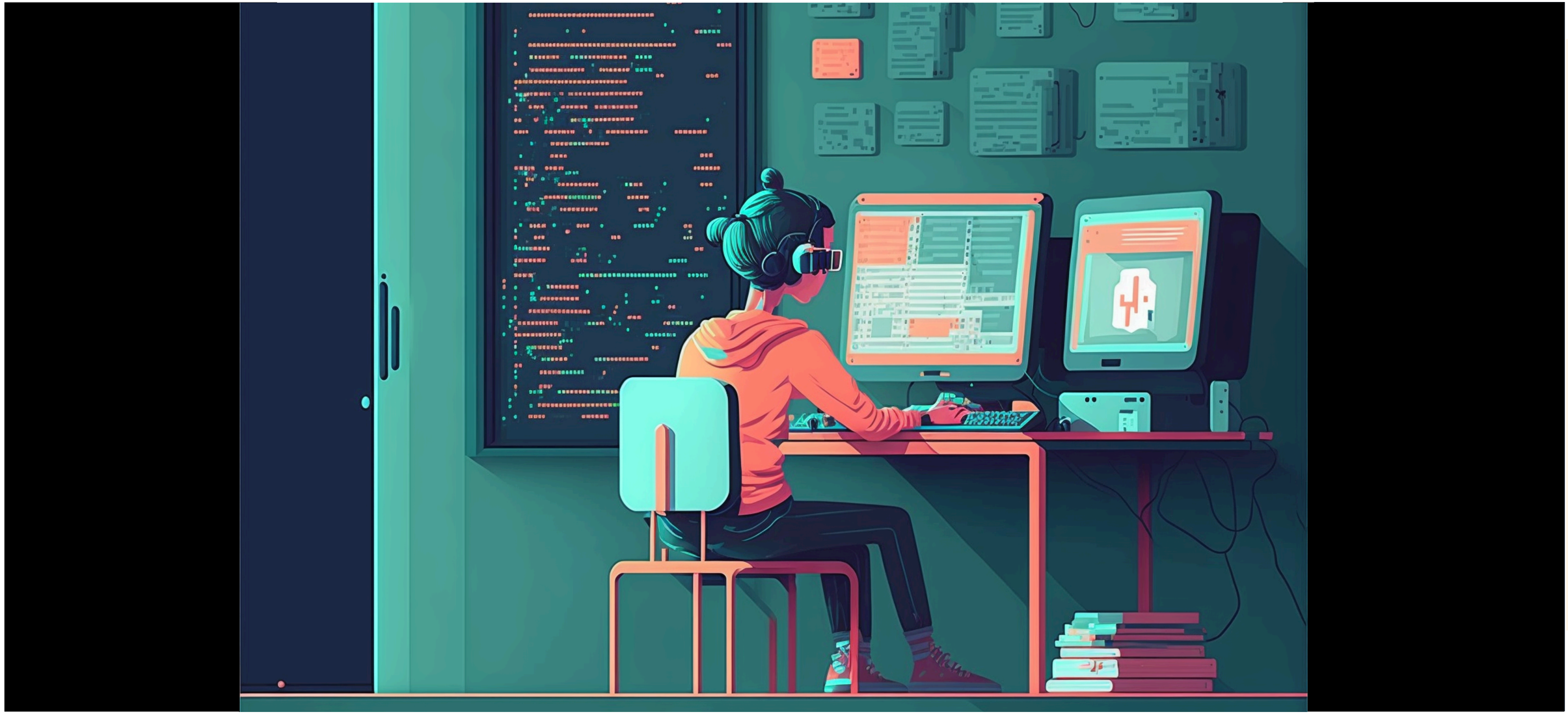


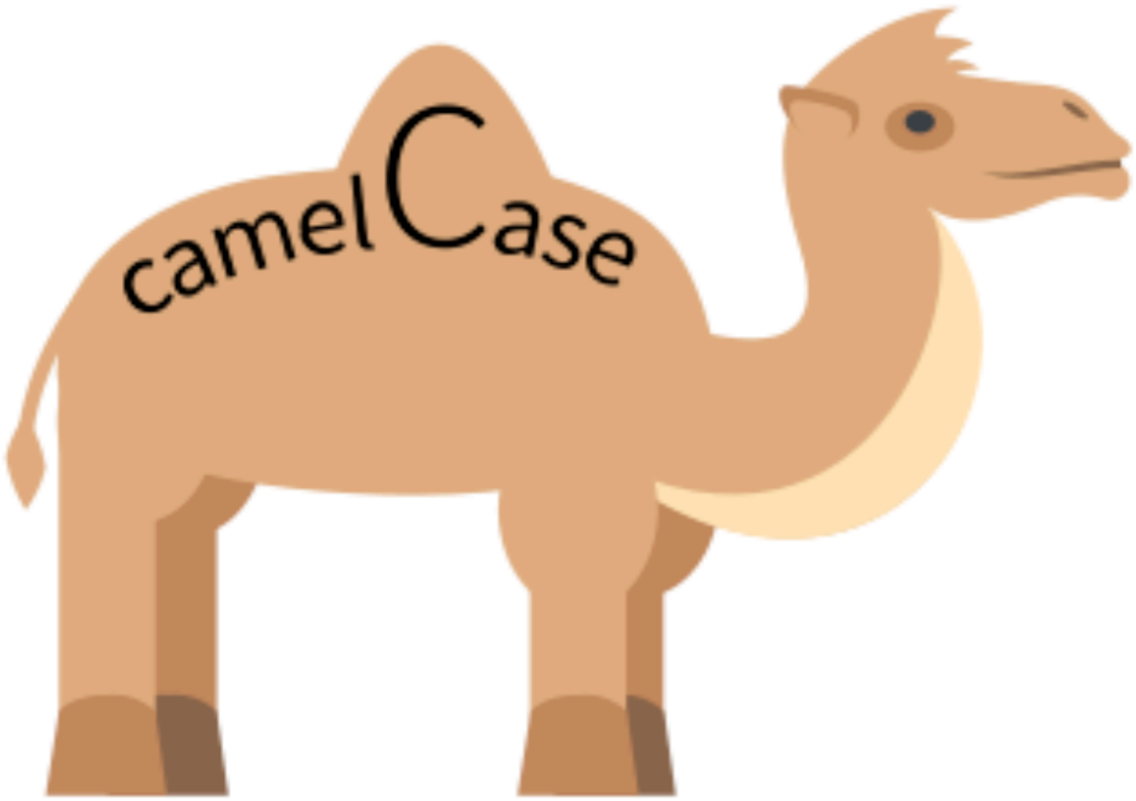
Mid Journey V4 prompt: a human student sitting in front of a desk learning programming language in a classroom --ar 3:2 --q 2

Lecture 1-3: Python Basics I

Dr. James Chen, Animal Data Scientist, School of Animal Sciences

Variables - Naming Convention

Camel case:	Snake case:
getName	get_name
buildBlock	build_block
setRule	set_rule



https://en.wikipedia.org/wiki/Camel_case

Variables - Naming Convention

styleguide

Google Python Style Guide

3.16 Naming

`module_name`, `package_name`, `ClassName`, `method_name`, `ExceptionName`, `function_name`, `GLOBAL_CONSTANT_NAME`, `global_var_name`, `instance_var_name`, `function_parameter_name`, `local_var_name`, `query_proper_noun_for_thing`, `send_acronym_via_https`.

3.16.1 Names to Avoid

- single character names, except for specifically allowed cases:
  - counters or iterators (e.g. `i`, `j`, `k`, `v`, et al.)
  - `e` as an exception identifier in `try/except` statements.
  - `f` as a file handle in `with` statements
  - private `TypeVar`s with no constraints (e.g. `_T`, `_U`, `_V`)

https://google.github.io/styleguide/pyguide.html

Variables - Naming Convention

styleguide

Google Python Style Guide

3.16 Naming

`module_name`, `package_name`, `ClassName`, `method_name`, `ExceptionName`, `function_name`, `GLOBAL_CONSTANT_NAME`, `global_var_name`, `instance_var_name`, `function_parameter_name`, `local_var_name`, `query_proper_noun_for_thing`, `send_acronym_via_https`.

3.16.1 Names to Avoid

- dashes (`-`) in any package/module name
- `__double_leading_and_trailing_underscore__` names (reserved by Python)
- offensive terms
- names that needlessly include the type of the variable (for example: `id_to_name_dict`)

https://google.github.io/styleguide/pyguide.html

Variables - Naming Convention

styleguide

Google Python Style Guide

3.16.4 Guidelines derived from Guido's Recommendations

Type	Public	Internal
Packages	`lower_with_under`	
Modules	`lower_with_under`	`_lower_with_under`
Classes	`CapWords`	`_CapWords`
Exceptions	`CapWords`	
Functions	`lower_with_under()`	`_lower_with_under()`
Global/Class Constants	`CAPS_WITH_UNDER`	`_CAPS_WITH_UNDER`
Global/Class Variables	`lower_with_under`	`_lower_with_under`
Instance Variables	`lower_with_under`	`_lower_with_under` (protected)
Method Names	`lower_with_under()`	`_lower_with_under()` (protected)
Function/Method Parameters	`lower_with_under`	
Local Variables	`lower_with_under`	

https://google.github.io/styleguide/pyguide.html

Variables - Naming Convention

1. Variables

There are several rules for naming variables in Python:

- A variable name can only contain letters (`A-Za-z`), numbers(`0-9`), and underscores(`_`).
- A variable name is case sensitive. For example, `first_var` and `First_var` are two different variables.
- There are two ways to name a variable: `snake_case` and `camelCase`. In `snake_case`, all letters are lowercase and words are separated by underscores. In `camelCase`, the first letter of each word is capitalized. For example, `first_var` and `firstVar` are both valid variable names.

Things you cannot do:

- A variable name cannot start with a number.
- A variable name cannot contain spaces.

Examples:

- Valid variable names: `first_var`, `firstVar`, `first_var_1`, `firstVar1`, `first_var_1_2_3`, `firstVar123`
- Invalid variable names: `1st_var`, `first var`, `first-var`

Variables - Assignment

```
first_var = 3
```

```
print(first_var) # 3
```

Data Types

There are several data types in Python:

- `int`: an integer number, e.g., `3`, `0`, `-1`
- `float`: a floating point number, e.g., `3.14`, `0.0`, `-1.0`
- `bool`: a boolean value, e.g., `True`, `False`
- `str`: a string, e.g., `"hello"`, `"2023-01-30"`

```
var_int = 3
var_float = 3.14
var_bool = True
var_str = "hello"
```

https://www.pytables.org/usersguide/datatypes.html

Data Types

Data types supported for array elements and tables columns in PyTables.

Type Code	Description	C Type	Size (in bytes)	Python Counterpart
bool	boolean	unsigned char	1	bool
int8	8-bit integer	signed char	1	int
uint8	8-bit unsigned integer	unsigned char	1	int
int16	16-bit integer	short	2	int
uint16	16-bit unsigned integer	unsigned short	2	int
int32	integer	int	4	int
uint32	unsigned integer	unsigned int	4	long
int64	64-bit integer	long long	8	long
uint64	unsigned 64-bit integer	unsigned long long	8	long
float16 [1]	half-precision float		2	
float32	single-precision float	float	4	float
float64	double-precision float	double	8	float
float96 [1] [2]	extended precision float		12	
float128 [1] [2]	extended precision float		16	
complex64	single-precision complex	struct {float r, i;}	8	complex
complex128	double-precision complex	struct {double r, i;}	16	complex
complex192 [1]	extended precision complex		24	
complex256 [1]	extended precision complex		32	
string	arbitrary length string	char[]		str
time32	integer time	POSIX's time_t	4	int
time64	floating point time	POSIX's struct timeval	8	float
enum	enumerated value	enum		

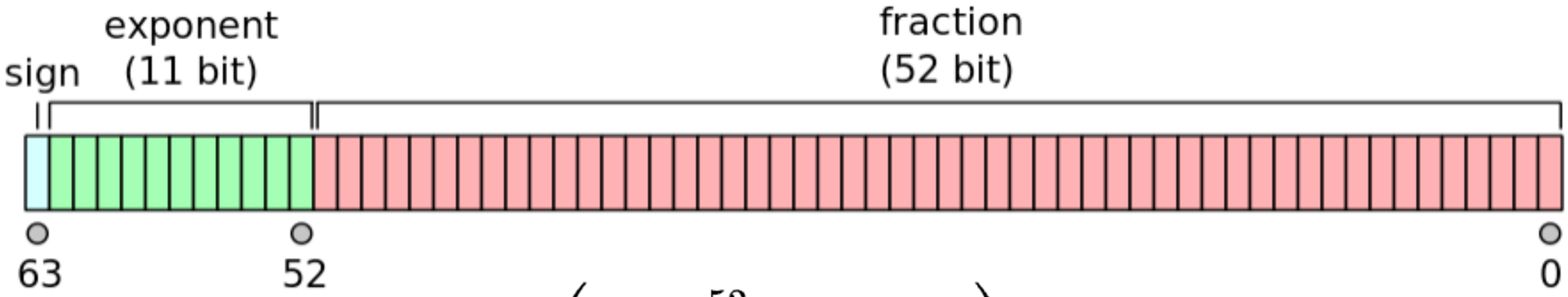
https://en.wikipedia.org/wiki/Double-precision_floating-point_format

Data Types - Int 8

Your note here:

https://en.wikipedia.org/wiki/Double-precision_floating-point_format

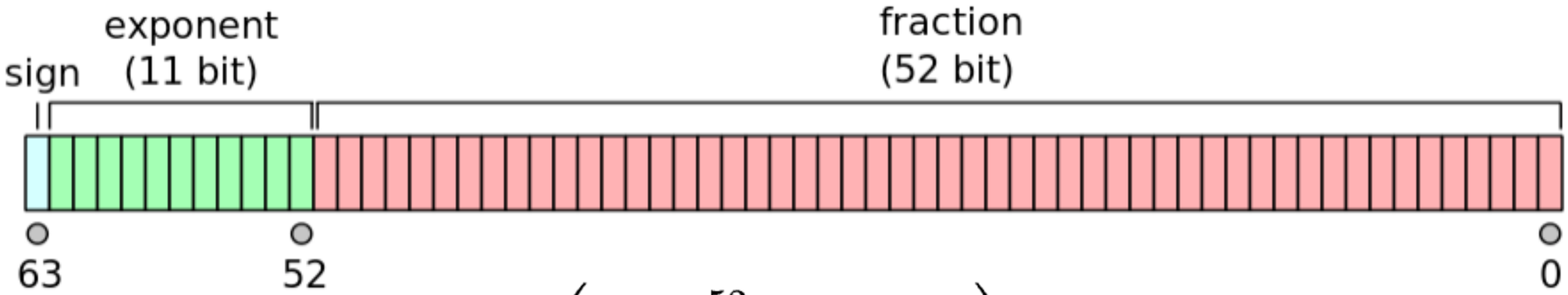
Data Types - Float64 (Double-Precision)



$$(-1)^{\text{sign}} \left(1 + \sum_{i=1}^{52} b_{52-i} 2^{-i}\right) \times 2^{e-1023}$$

https://en.wikipedia.org/wiki/Double-precision_floating-point_format

Data Types - Float64 (Double-Precision)

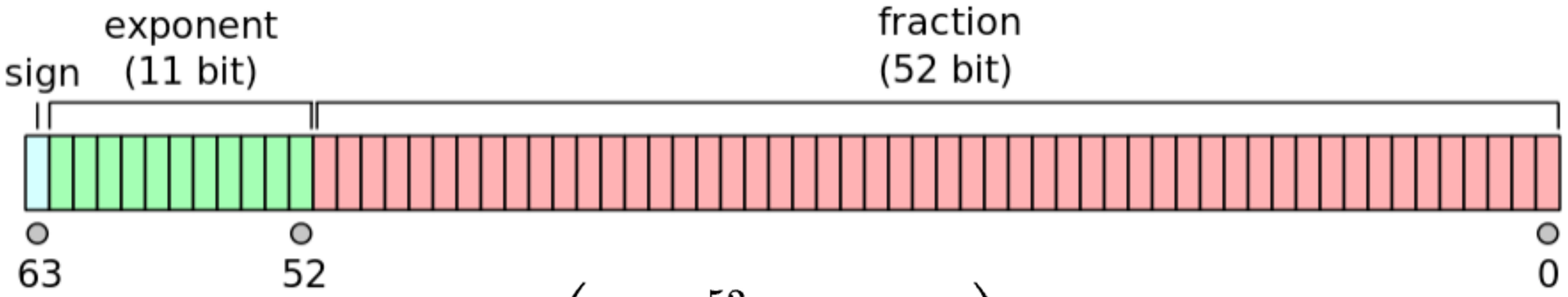


$$(-1)^{\text{sign}} \left(1 + \sum_{i=1}^{52} b_{52-i} 2^{-i}\right) \times 2^{e-1023}$$

$5 \leftarrow +2^2 \times 1.01_2$

https://en.wikipedia.org/wiki/Double-precision_floating-point_format

Data Types - Float64 (Double-Precision)



$$(-1)^{\text{sign}} \left(1 + \sum_{i=1}^{52} b_{52-i} 2^{-i}\right) \times 2^{e-1023}$$

1	$+2^0 \times 1$	0 01111111111 0000000000000000000000000000000000000000000000000000$_2$
2	$+2^1 \times 1$	0 10000000000 0000000000000000000000000000000000000000000000000000$_2$
-2	$-2^1 \times 1$	1 10000000000 0000000000000000000000000000000000000000000000000000$_2$
3	$+2^1 \times 1.1_2$	0 10000000000 1000000000000000000000000000000000000000000000000000$_2$
4	$+2^2 \times 1$	0 10000000001 0000000000000000000000000000000000000000000000000000$_2$
5	$+2^2 \times 1.01_2$	0 10000000001 0100000000000000000000000000000000000000000000000000$_2$
6	$+2^2 \times 1.1_2$	0 10000000001 1000000000000000000000000000000000000000000000000000$_2$

https://cloud.google.com/tpu/docs/bfloat16

Data Types - Floating Point Formats

Floating Point Formats

bfloat16: Brain Floating Point Format

Range: ~$1e^{-38}$ to ~$3e^{38}$

fp32: Single-precision IEEE Floating Point Format

Range: ~$1e^{-38}$ to ~$3e^{38}$

fp16: Half-precision IEEE Floating Point Format

Range: ~$5.96e^{-8}$ to 65504

Data Types - Type Casting

From `float` to `int`:

```
var_float = 3.14
var_int = int(var_float)
print(var_int) # 3
```

From `int` to `string`:

```
var_int = 3
var_str = str(var_int)
print(var_str) # "3"
```

Operators - Arithmetic Operators

```
a = 7
b = 4
```

Operator	Description	Example	Result
`+`	Addition	`a + b`	`11`
`-`	Subtraction	`a - b`	`3`
`*`	Multiplication	`a * b`	`28`
`**`	Exponentiation	`a ** b`	`2401`
`/`	Division	`a / b`	`1.75`
`//`	Floor division	`a // b`	`1`
`%`	Modulus	`a % b`	`3`

Operators - Comparison Operators

```
a = 7
b = 4
```

Operator	Description	Example	Result
`==`	Equal to	`a == b`	`False`
`!=`	Not equal to	`a != b`	`True`
`>`	Greater than	`a > b`	`True`
`<`	Less than	`a < b`	`False`
`>=`	Greater than or equal to	`a >= b`	`True`
`<=`	Less than or equal to	`a <= b`	`False`

Operators - String Operators

```
a = "hello"
b = "world"
```

Operator	Description	Example	Result
`+`	Concatenation	`a + b`	`"helloworld"`
`*`	Repetition	`a * 3`	`"hellohellohello"`

Operators - String Operators

```
a = "hello"
b = "world"
```

How to we compare two strings?

Operators - String Operators (ASCII Table)

```
a = "hello"
b = "world"
```

How to we compare two strings?

Every character can be represented by an ASCII code (integer)

Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char
0	0	[NULL]	32	20	[SPACE]	64	40	@	96	60	`
1	1	[START OF HEADING]	33	21	!	65	41	A	97	61	a
2	2	[START OF TEXT]	34	22	"	66	42	B	98	62	b
3	3	[END OF TEXT]	35	23	#	67	43	C	99	63	c
4	4	[END OF TRANSMISSION]	36	24	$	68	44	D	100	64	d
5	5	[ENQUIRY]	37	25	%	69	45	E	101	65	e
6	6	[ACKNOWLEDGE]	38	26	&	70	46	F	102	66	f
7	7	[BELL]	39	27	'	71	47	G	103	67	g
8	8	[BACKSPACE]	40	28	(	72	48	H	104	68	h
9	9	[HORIZONTAL TAB]	41	29	)	73	49	I	105	69	i
10	A	[LINE FEED]	42	2A	*	74	4A	J	106	6A	j
11	B	[VERTICAL TAB]	43	2B	+	75	4B	K	107	6B	k
12	C	[FORM FEED]	44	2C	,	76	4C	L	108	6C	l
13	D	[CARRIAGE RETURN]	45	2D	-	77	4D	M	109	6D	m
14	E	[SHIFT OUT]	46	2E	.	78	4E	N	110	6E	n
15	F	[SHIFT IN]	47	2F	/	79	4F	O	111	6F	o
16	10	[DATA LINK ESCAPE]	48	30	0	80	50	P	112	70	p
17	11	[DEVICE CONTROL 1]	49	31	1	81	51	Q	113	71	q
18	12	[DEVICE CONTROL 2]	50	32	2	82	52	R	114	72	r
19	13	[DEVICE CONTROL 3]	51	33	3	83	53	S	115	73	s
20	14	[DEVICE CONTROL 4]	52	34	4	84	54	T	116	74	t
21	15	[NEGATIVE ACKNOWLEDGE]	53	35	5	85	55	U	117	75	u
22	16	[SYNCHRONOUS IDLE]	54	36	6	86	56	V	118	76	v
23	17	[END OF TRANS. BLOCK]	55	37	7	87	57	W	119	77	w
24	18	[CANCEL]	56	38	8	88	58	X	120	78	x
25	19	[END OF MEDIUM]	57	39	9	89	59	Y	121	79	y
26	1A	[SUBSTITUTE]	58	3A	:	90	5A	Z	122	7A	z
27	1B	[ESCAPE]	59	3B	;	91	5B	[	123	7B	{
28	1C	[FILE SEPARATOR]	60	3C	<	92	5C	\	124	7C	|
29	1D	[GROUP SEPARATOR]	61	3D	=	93	5D	]	125	7D	}
30	1E	[RECORD SEPARATOR]	62	3E	>	94	5E	^	126	7E	~
31	1F	[UNIT SEPARATOR]	63	3F	?	95	5F	_	127	7F	[DEL]

https://upload.wikimedia.org/wikipedia/commons/thumb/1/1b/ASCII-Table-wide.svg/1200px-ASCII-Table-wide.svg.png

Operators - String Operators (ASCII Table)

```
a = "hello"
b = "world"
```

How to we compare two strings?

Every character can be represented by an ASCII code (integer)

Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char
0	0	[NULL]	32	20	[SPACE]	64	40	@	96	60	`
1	1	[START OF HEADING]	33	21	!	65	41	A	97	61	a
2	2	[START OF TEXT]	34	22	"	66	42	B	98	62	b
3	3	[END OF TEXT]	35	23	#	67	43	C	99	63	c
4	4	[END OF TRANSMISSION]	36	24	$	68	44	D	100	64	d
5	5	[ENQUIRY]	37	25	%	69	45	E	101	65	e
6	6	[ACKNOWLEDGE]	38	26	&	70	46	F	102	66	f
7	7	[BELL]	39	27	'	71	47	G	103	67	g
8	8	[BACKSPACE]	40	28	(	72	48	H	104	68	h
9	9	[HORIZONTAL TAB]	41	29	)	73	49	I	105	69	i
10	A	[LINE FEED]	42	2A	*	74	4A	J	106	6A	j
11	B	[VERTICAL TAB]	43	2B	+	75	4B	K	107	6B	k
12	C	[FORM FEED]	44	2C	,	76	4C	L	108	6C	l
13	D	[CARRIAGE RETURN]	45	2D	-	77	4D	M	109	6D	m
14	E	[SHIFT OUT]	46	2E	.	78	4E	N	110	6E	n
15	F	[SHIFT IN]	47	2F	/	79	4F	O	111	6F	o
16	10	[DATA LINK ESCAPE]	48	30	0	80	50	P	112	70	p
17	11	[DEVICE CONTROL 1]	49	31	1	81	51	Q	113	71	q
18	12	[DEVICE CONTROL 2]	50	32	2	82	52	R	114	72	r
19	13	[DEVICE CONTROL 3]	51	33	3	83	53	S	115	73	s
20	14	[DEVICE CONTROL 4]	52	34	4	84	54	T	116	74	t
21	15	[NEGATIVE ACKNOWLEDGE]	53	35	5	85	55	U	117	75	u
22	16	[SYNCHRONOUS IDLE]	54	36	6	86	56	V	118	76	v
23	17	[END OF TRANS. BLOCK]	55	37	7	87	57	W	119	77	w
24	18	[CANCEL]	56	38	8	88	58	X	120	78	x
25	19	[END OF MEDIUM]	57	39	9	89	59	Y	121	79	y
26	1A	[SUBSTITUTE]	58	3A	:	90	5A	Z	122	7A	z
27	1B	[ESCAPE]	59	3B	;	91	5B	[	123	7B	{
28	1C	[FILE SEPARATOR]	60	3C	<	92	5C	\	124	7C	|
29	1D	[GROUP SEPARATOR]	61	3D	=	93	5D	]	125	7D	}
30	1E	[RECORD SEPARATOR]	62	3E	>	94	5E	^	126	7E	~
31	1F	[UNIT SEPARATOR]	63	3F	?	95	5F	_	127	7F	[DEL]

https://upload.wikimedia.org/wikipedia/commons/thumb/1/1b/ASCII-Table-wide.svg/1200px-ASCII-Table-wide.svg.png

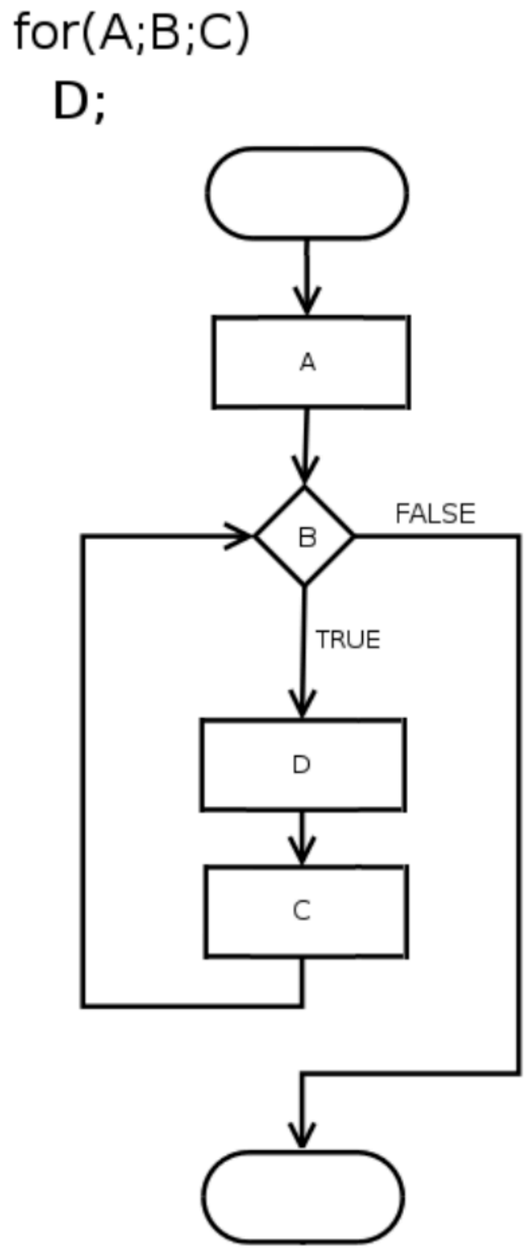
```
a = "hello"
b = "world"
print(a != b) # True
print(a > b) # False
```

Operators - Logical Operators

```
a = True
b = False
```

Operator	Description	Example	Result
`and`	Logical AND	`a and b`	`False`
&&	Logical AND	`a && b`	`False`
`or`	Logical OR	`a or b`	`True`
||	Logical OR	`a || b`	`True`
`not`	Logical NOT	`not a`	`False`
!	Logical NOT	`!a`	`False`

Flow Control Statement - If Statement

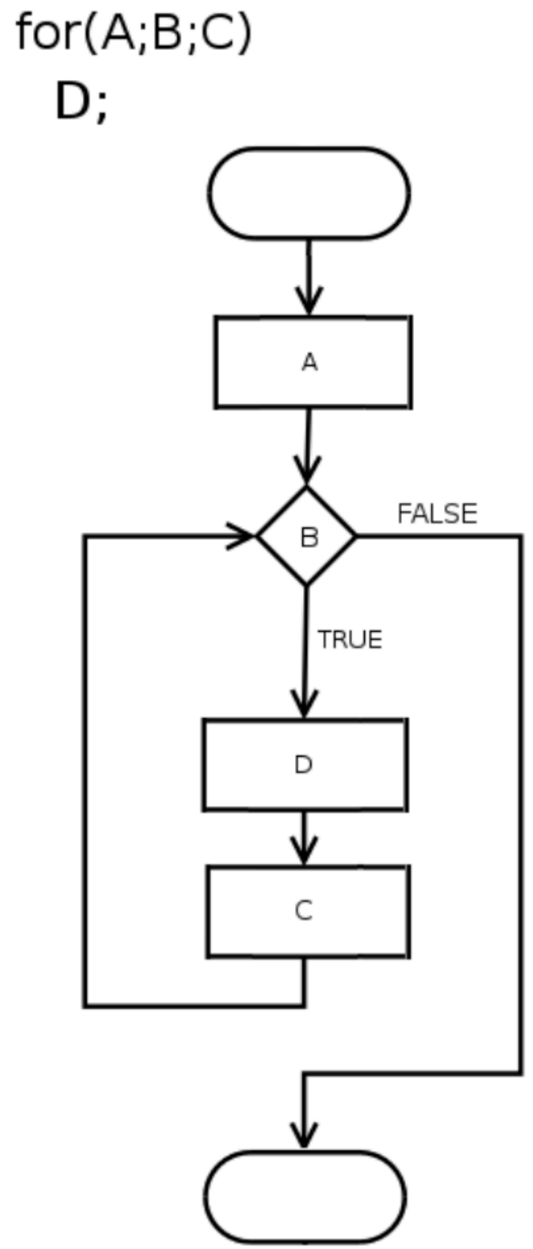


https://en.wikipedia.org/wiki/Control_flow

Flow Control Statement - If Statement

```
if 10 > 0:
    print("10 is greater than 0")

# Output:
# 10 is greater than 0
```



https://en.wikipedia.org/wiki/Control_flow

Flow Control Statement - If Statement

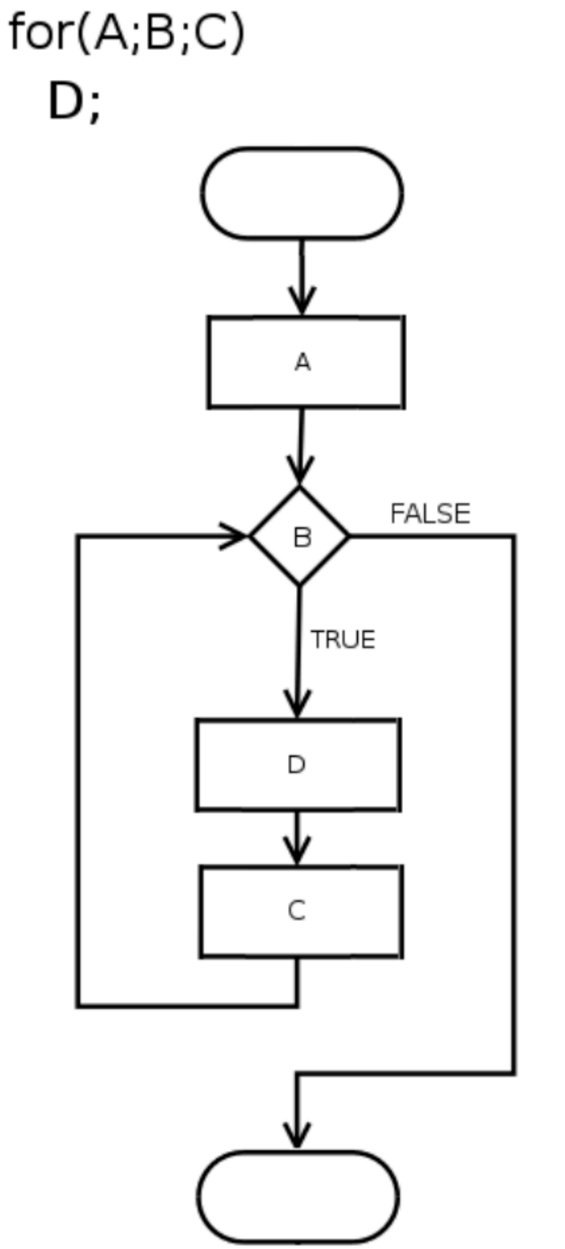
```
if 10 > 0:
    print("10 is greater than 0")

# Output:
# 10 is greater than 0
```

Indentation is important in Python!

```
if 10 > 0:
print("10 is greater than 0")

# Output:
# IndentationError: expected an indented block
```

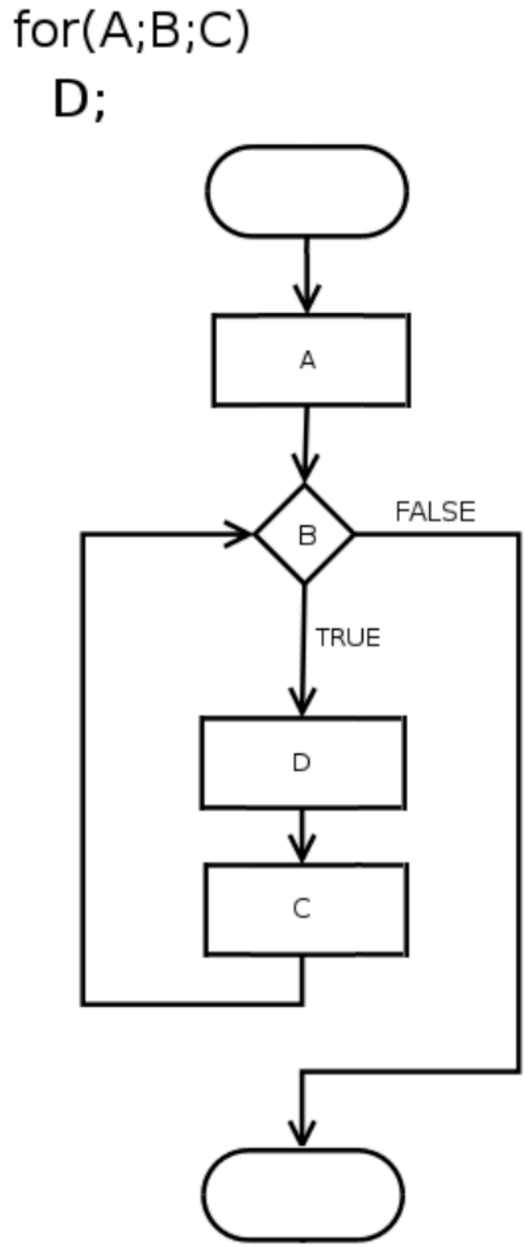


https://en.wikipedia.org/wiki/Control_flow

Flow Control Statement - If-Else Statement

```
if 7 % 2 == 0:
    print("7 is even")
else:
    print("7 is odd")

# Output:
# 7 is odd
```

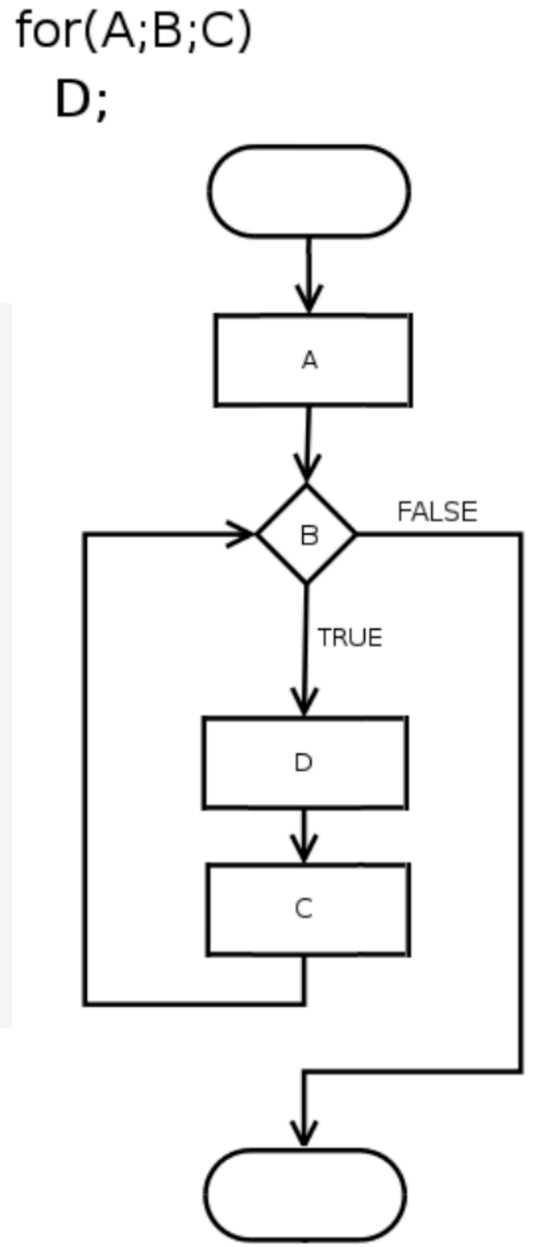


https://en.wikipedia.org/wiki/Control_flow

Flow Control Statement - If-Elif-Else Statement

```
if 27 % 2 == 0:
    print("27 is even")
elif 27 % 3 == 0:
    print("27 is divisible by 3")
else:
    print("27 is neither even nor divisible by 3")

# Output:
# 27 is divisible by 3
```



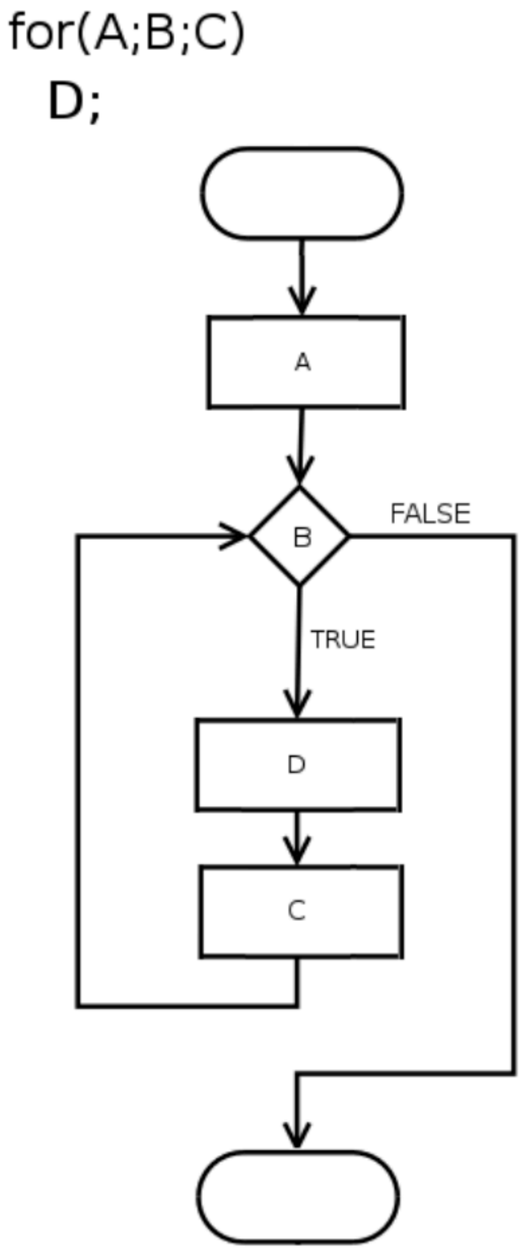
https://en.wikipedia.org/wiki/Control_flow

Flow Control Statement - Ternary operator

```
value = ""
if 10 > 0:
    value = "10 is greater than 0"
else:
    value = "10 is not greater than 0"
```

Which is equivalent to:

```
value = "10 is greater than 0" if 10 > 0 else "10 is not greater than 0"
```



https://en.wikipedia.org/wiki/Control_flow