به نام خدا

مبانی رایانه

(مروری بر انواع داده در زبان پایتون ۳)

حجت قلی‌زاده و مجتبی اعلائی

دانشگاه صنعتی اصفهان

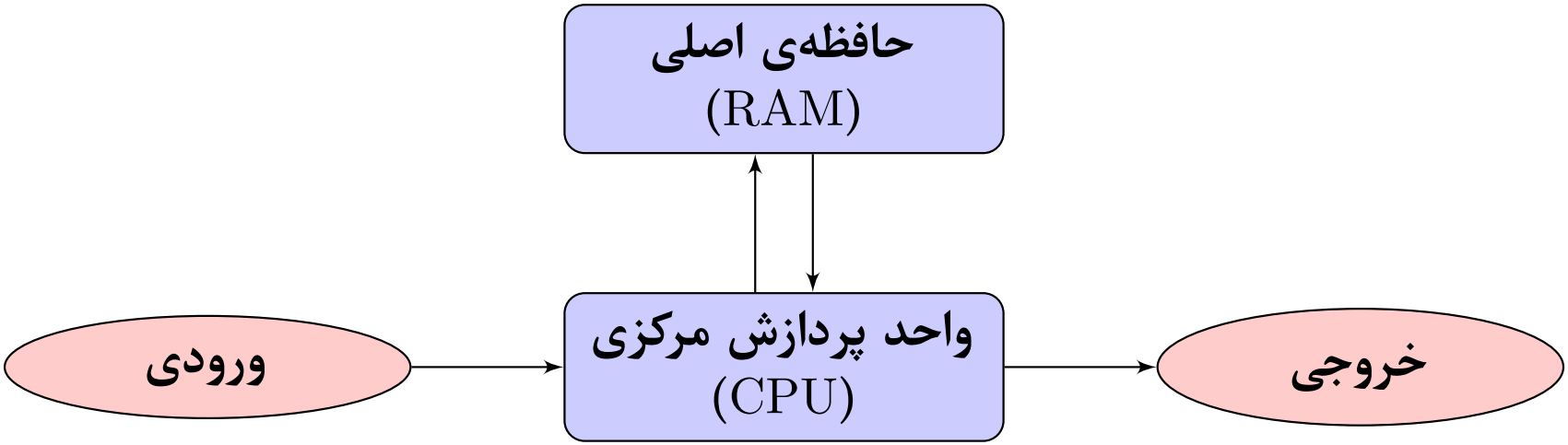
دانشکده‌ی فیزیک

۱۶ بهمن‌ماه ۱۳۹۶، دانشگاه صنعتی اصفهان

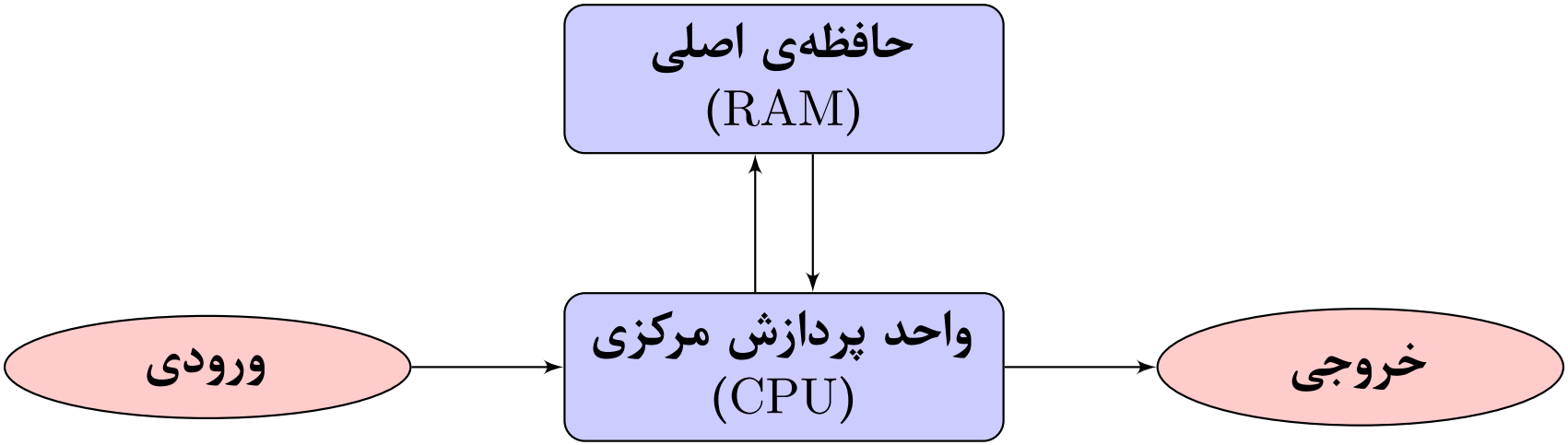
عناوین

عناوین

پیکربندی عمومی رایانه



پیکربندی عمومی رایانه



صفحه‌کلید
ماوس
دوربین
میکروفون
انواع حسگرها
...
پرونده (فایل)

نمایشگر
بلندگو
چاپگر
...
پرونده (فایل)

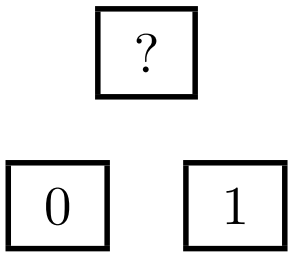
عناوین

حافظه‌ی اصلی رایانه: ساختمان فیزیکی

در رایانه‌های الکترونیکی، حافظه عبارت است از دنباله‌ای از یاخته‌های با قابلیت نگهداری یکی از مقادیر ۰ یا ۱. هر یک از این یاخته‌ها یک بیت (bit) نامیده می‌شود.

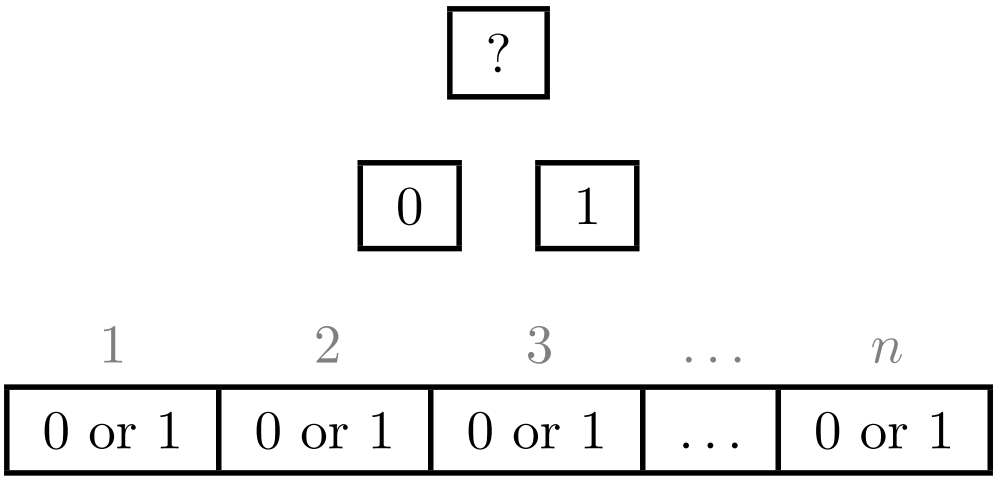
حافظه‌ی اصلی رایانه: ساختمان فیزیکی

در رایانه‌های الکترونیکی، حافظه عبارت است از دنباله‌ای از یاخته‌های با قابلیت نگهداری یکی از مقادیر ۰ یا ۱. هر یک از این یاخته‌ها یک بیت (bit) نامیده می‌شود.



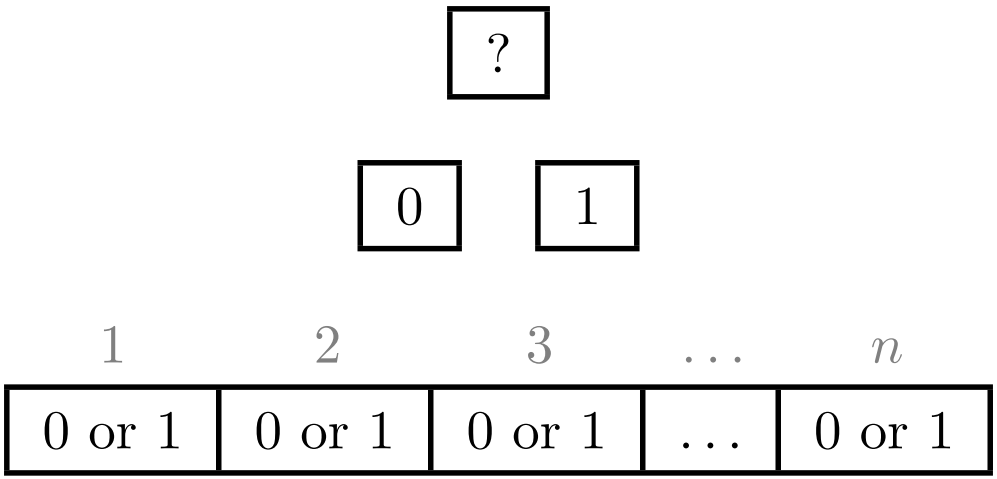
حافظه‌ی اصلی رایانه: ساختمان فیزیکی

در رایانه‌های الکترونیکی، حافظه عبارت است از دنباله‌ای از یاخته‌های با قابلیت نگهداری یکی از مقادیر ۰ یا ۱. هر یک از این یاخته‌ها یک بیت (bit) نامیده می‌شود.



حافظه‌ی اصلی رایانه: ساختمان فیزیکی

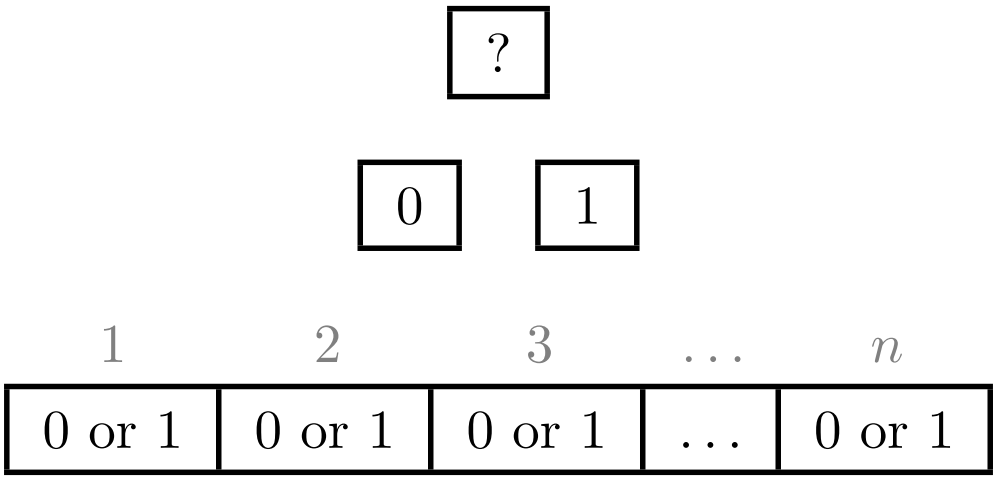
در رایانه‌های الکترونیکی، حافظه عبارت است از دنباله‌ای از یاخته‌های با قابلیت نگهداری یکی از مقادیر ۰ یا ۱. هر یک از این یاخته‌ها یک بیت (bit) نامیده می‌شود.



دنباله‌ای از n بیت امکان نمایش ۲^n حالت متمایز را فراهم می‌کند.

حافظه‌ی اصلی رایانه: ساختمان فیزیکی

در رایانه‌های الکترونیکی، حافظه عبارت است از دنباله‌ای از یاخته‌های با قابلیت نگهداری یکی از مقادیر ۰ یا ۱. هر یک از این یاخته‌ها یک بیت (bit) نامیده می‌شود.



دنباله‌ای از n بیت امکان نمایش ۲^n حالت متمایز را فراهم می‌کند.

یک دنباله‌ی ۸ بیتی یک بایت (bite) نامیده می‌شود. یک بایت می‌تواند $۲۵۶ = ۲^۸$ حالت متمایز را نمایش دهد:

00000000, 00000001, 00000010, 00000011, 00000100, ..., 11111101, 11111110, 11111111

حافظه‌ی اصلی رایانه: ساختمان فیزیکی

- هر بایت از حافظه‌ی رایانه دارای یک نشانی (address) است که با یک عدد صحیح غیرمنفی نمایش داده می‌شود. مثلاً، بایت شماره‌ی ۰، بایت شماره‌ی ۱، بایت شماره‌ی ۲، .... از لحاظ نظری، بیشینه‌ی این عدد صحیح حد بالای میزان حافظه‌ی قابل پشتیبانی با هر رایانه را تعیین می‌کند.
- در رایانه‌های الکترونیکی ۱۶ بیتی (۱۹۵۱)، نشانی هر بایت از حافظه با یک دنباله‌ی ۱۶ بیتی (یک عدد دودویی ۱۶ رقمی) نمایش داده می‌شود. بنابراین، چنین رایانه‌ای می‌تواند حداکثر $۶۵‘۵۳۶ = ۲^{۱۶}$ بایت (۶۴ کیلوبایت (KiB)) حافظه را پشتیبانی کند.
- در رایانه‌های الکترونیکی ۳۲ بیتی (۱۹۸۵)، نشانی هر بایت از حافظه با یک دنباله‌ی ۳۲ بیتی (یک عدد دودویی ۳۲ رقمی) نمایش داده می‌شود. بنابراین، چنین رایانه‌ای می‌تواند حداکثر $۴‘۲۹۴‘۹۶۷‘۲۹۶ = ۲^{۳۲}$ بایت (۴ گیگابایت (GiB)) حافظه را پشتیبانی کند.
- در رایانه‌های الکترونیکی ۶۴ بیتی (۲۰۰۳)، نشانی هر بایت از حافظه با یک دنباله‌ی ۶۴ بیتی (یک عدد دودویی ۶۴ رقمی) نمایش داده می‌شود. بنابراین، چنین رایانه‌ای می‌تواند حداکثر $۱۸‘۴۴۶‘۷۴۴‘۰۷۳‘۷۰۹‘۵۵۱‘۶۱۶ = ۲^{۶۴}$ بایت (۱۶ اگزابایت (EiB)) حافظه را پشتیبانی کند.

عناوین

انواع داده‌های درونی: متن

به منظور ذخیره‌ی متن در حافظه‌ی یارانه، تناظری یک‌به‌یک بین هر نویسه (character) از متن و رشته‌ای از بیت‌ها لازم است.

- در سال ۱۹۶۳، یک نسخه‌ی ارتقا یافته از استاندارد های موجود برای ارسال متون انگلیسی ساده در سامانه‌ی تلگراف ارائه شد. این استاندارد جدید که شامل حداکثر ۱۲۸ نویسه (قابل نمایش با ۷ بیت) بود، American Standard Code for Information Interchange و به اختصار ASCII نامیده می‌شود. به عبارت دیگر، استاندارد اَسکی تناظری یک‌به‌یک بین نویسه‌های انگلیسی متداول و اعداد صحیح در بازه‌ی ۰ تا ۱۲۷ (۷ بیت) ایجاد می‌کند.
- همان طور که ذکر شد، یک بایت می‌تواند ۲۵۶ حالت متفاوت (متناظر با اعداد صحیح ۰ تا ۲۵۵) را نمایش دهد. در سال‌های بعد، به منظور پشتیبانی از نویسه‌های سایر زبان‌ها، با استفاده از بیت هشتم هر بایت، این استاندارد به extended ASCII (EASCII) توسعه داده شد. استاندارد EASCII دارای بخش‌هایی متفاوت برای زبان‌های مختلف بود. به عبارت دیگر، نویسه‌های ۰ تا ۱۲۷ در همه‌ی بخش‌ها یکسان بودند، اما نویسه‌های ۱۲۸ تا ۲۵۵ برای زبان‌های مختلف متفاوت بودند. در نتیجه، امکان نوشتن متون چند زبانه وجود نداشت.
- در نهایت، به منظور پشتیبانی از همه‌ی زبان‌ها در یک استاندارد واحد، استاندارد یونیکُد با استفاده از حداکثر ۴ بایت (۳۲ بیت) و قابلیت نمایش ۱,۱۱۴,۱۱۲ نویسه توسعه داده شد.

انواع داده‌های درونی: نویسه‌های اَسکی

Dec	Hex	Oct	Chr	Dec	Hex	Oct	HTML	Chr	Dec	Hex	Oct	HTML	Chr	Dec	Hex	Oct	HTML	Chr
0	0	000	NULL	32	20	040	&#032;	Space	64	40	100	&#064;	@	96	60	140	&#096;	`
1	1	001	Start of Header	33	21	041	&#033;	!	65	41	101	&#065;	A	97	61	141	&#097;	a
2	2	002	Start of Text	34	22	042	&#034;	"	66	42	102	&#066;	B	98	62	142	&#098;	b
3	3	003	End of Text	35	23	043	&#035;	#	67	43	103	&#067;	C	99	63	143	&#099;	c
4	4	004	End of Transmission	36	24	044	&#036;	$	68	44	104	&#068;	D	100	64	144	&#100;	d
5	5	005	Enquiry	37	25	045	&#037;	%	69	45	105	&#069;	E	101	65	145	&#101;	e
6	6	006	Acknowledgment	38	26	046	&#038;	&	70	46	106	&#070;	F	102	66	146	&#102;	f
7	7	007	Bell	39	27	047	&#039;	'	71	47	107	&#071;	G	103	67	147	&#103;	g
8	8	010	Backspace	40	28	050	&#040;	(	72	48	110	&#072;	H	104	68	150	&#104;	h
9	9	011	Horizontal Tab	41	29	051	&#041;	)	73	49	111	&#073;	I	105	69	151	&#105;	i
10	A	012	Line feed	42	2A	052	&#042;	*	74	4A	112	&#074;	J	106	6A	152	&#106;	j
11	B	013	Vertical Tab	43	2B	053	&#043;	+	75	4B	113	&#075;	K	107	6B	153	&#107;	k
12	C	014	Form feed	44	2C	054	&#044;	,	76	4C	114	&#076;	L	108	6C	154	&#108;	l
13	D	015	Carriage return	45	2D	055	&#045;	-	77	4D	115	&#077;	M	109	6D	155	&#109;	m
14	E	016	Shift Out	46	2E	056	&#046;	.	78	4E	116	&#078;	N	110	6E	156	&#110;	n
15	F	017	Shift In	47	2F	057	&#047;	/	79	4F	117	&#079;	O	111	6F	157	&#111;	o
16	10	020	Data Link Escape	48	30	060	&#048;	0	80	50	120	&#080;	P	112	70	160	&#112;	p
17	11	021	Device Control 1	49	31	061	&#049;	1	81	51	121	&#081;	Q	113	71	161	&#113;	q
18	12	022	Device Control 2	50	32	062	&#050;	2	82	52	122	&#082;	R	114	72	162	&#114;	r
19	13	023	Device Control 3	51	33	063	&#051;	3	83	53	123	&#083;	S	115	73	163	&#115;	s
20	14	024	Device Control 4	52	34	064	&#052;	4	84	54	124	&#084;	T	116	74	164	&#116;	t
21	15	025	Negative Ack.	53	35	065	&#053;	5	85	55	125	&#085;	U	117	75	165	&#117;	u
22	16	026	Synchronous idle	54	36	066	&#054;	6	86	56	126	&#086;	V	118	76	166	&#118;	v
23	17	027	End of Trans. Block	55	37	067	&#055;	7	87	57	127	&#087;	W	119	77	167	&#119;	w
24	18	030	Cancel	56	38	070	&#056;	8	88	58	130	&#088;	X	120	78	170	&#120;	x
25	19	031	End of Medium	57	39	071	&#057;	9	89	59	131	&#089;	Y	121	79	171	&#121;	y
26	1A	032	Substitute	58	3A	072	&#058;	:	90	5A	132	&#090;	Z	122	7A	172	&#122;	z
27	1B	033	Escape	59	3B	073	&#059;	;	91	5B	133	&#091;	[	123	7B	173	&#123;	{
28	1C	034	File Separator	60	3C	074	&#060;	<	92	5C	134	&#092;	\	124	7C	174	&#124;	|
29	1D	035	Group Separator	61	3D	075	&#061;	=	93	5D	135	&#093;	]	125	7D	175	&#125;	}
30	1E	036	Record Separator	62	3E	076	&#062;	>	94	5E	136	&#094;	^	126	7E	176	&#126;	~
31	1F	037	Unit Separator	63	3F	077	&#063;	?	95	5F	137	&#095;	_	127	7F	177	&#127;	Del

asciichars.com

انواع داده‌های درونی: اعداد صحیح

اعداد صحیح (integer) پس از تبدیل مبنا از ۱۰ (decimal) به ۲ (binary)، به سادگی در دنباله‌ای از بیت‌ها ذخیره می‌شوند:

مبنای ۱۰	مبنای ۲
0	0
1	1
2	10
3	11
4	100
5	101
6	110
7	111

مبنای ۱۰	مبنای ۲
8	1000
9	1001
10	1010
11	1011
12	1100
13	1101
14	1110
15	1111

مبنای ۱۰	مبنای ۲
16	10000
17	10001
18	10010
19	10011
20	10100
21	10101
22	10110
23	10111
⋮	⋮

تعداد ارقام لازم برای نمایش عدد طبیعی N در مبنای b عبارت است از:

$$n = [\log_b N] + ۱ = [\frac{\ln N}{\ln b}] + ۱. \quad (۱)$$

انواع داده‌های درونی: اعداد صحیح

زبان‌های برنامه‌نویسی معمولاً چند نوع از اعداد صحیح با بازه‌های متفاوت را پشتیبانی می‌کنند.

تعداد بایت	تعداد بیت	بازه	C	Fortran	Python
1	8	$-128 \ldots +127$	`signed char`	`INT(KIND=1)`	`numpy.int8`
1	8	$0 \ldots +255$	`unsigned char`	———	`numpy.uint8`
2	16	$-32{,}768 \ldots +32{,}767$	`signed short`	`INT(KIND=2)`	`numpy.int16`
2	16	$0 \ldots +65{,}536$	`unsigned short`	———	`numpy.uint16`
4	32	$-2{,}147{,}483{,}648 \ldots +2{,}147{,}483{,}647$	`signed long`	`INT(KIND=4)`	`numpy.int32`
4	32	$0 \ldots +4{,}294{,}967{,}295$	`unsigned long`	———	`numpy.uint32`
8	64	$-9{,}223{,}372{,}036{,}854{,}775{,}808$ $\ldots +9{,}223{,}372{,}036{,}854{,}775{,}807$	`signed long long`	`INT(KIND=8)`	`numpy.int64`
8	64	$0 \ldots +18{,}446{,}744{,}073{,}709{,}551{,}615$	`unsigned long long`	———	`numpy.uint64`
نامحدود	نامحدود	نامحدود	———	———	`int`

به دلیل تفاوت در ساختار زبان‌های مختلف، در صورتی که مقدار عدد صحیح خارج از بازه‌ی آن قرار گیرد ممکن است انواع متفاوتی از خطا پیش آید.

انواع داده‌های درونی: اعداد صحیح

عملیات ریاضی و مقایسه‌ی اعداد صحیح سرراست است:

```
>>> m = numpy.int32(2147483647)
>>> m
2147483647
>>>
>>> n = m + numpy.int32(1)
>>> n
-2147483648
>>>
>>>
>>> a = 12345678901234567890123456789012345678901234567890
>>> a
12345678901234567890123456789012345678901234567890
>>>
>>> a + 1
12345678901234567890123456789012345678901234567891
>>>
>>> a + 1 > a
True
```

انواع داده‌های درونی: اعداد اعشاری

اعداد اعشاری (float) را می‌توان به دو شیوه در حافظه‌ی رایانه ذخیره کرد:

۱. **اعداد اعشاری با ممیز ثابت (fixed-point float numbers):**
در این روش همواره تعدادی ثابت از ارقام **قبل و بعد از ممیز** نگه داشته می‌شوند. به عبارت دیگر، از دو عدد صحیح با تعداد ارقام m و n به ترتیب برای ذخیره‌ی ارقام قبل و بعد از ممیز استفاده می‌گردد:

$$f = d_1 d_2 d_3 \dots d_m \bullet d_1 d_2 d_3 \dots d_n$$

۲. **اعداد اعشاری با ممیز شناور (floating-point float numbers):**
در این روش همواره تعدادی ثابت از ارقام **با معنی** نگه داشته می‌شوند. به عبارت دیگر، از یک عدد صحیح با تعداد ارقام m برای ذخیره‌ی ارقام بامعنی و از یک عدد صحیح دیگر برای ذخیره‌ی مرتبه‌ی بزرگی استفاده می‌گردد:

$$f = d_1 d_2 d_3 \dots d_m \times 10^n$$

انواع داده‌های درونی: اعداد اعشاری با ممیز ثابت

اعداد اعشاری با ممیز ثابت (fixed-point float numbers):
در این روش همواره تعدادی ثابت از ارقام **قبل و بعد از ممیز** نگه داشته می‌شوند. به عبارت دیگر، از دو عدد صحیح با تعداد ارقام m و n به ترتیب برای ذخیره‌ی ارقام قبل و بعد از ممیز استفاده می‌گردد:

$$f = d_1 d_2 d_3 \dots d_m \bullet d_1 d_2 d_3 \dots d_n$$

مهم‌ترین ویژگی‌های این روش عبارتند از:

- کوچک‌ترین و بزرگ‌ترین اعداد مثبت قابل نمایش در این روش به ترتیب برابرند با 10^{-n} و $10^m - 10^{-n} \approx 10^m$.
- نقطه‌ی قوت این روش دقت بالای آن در بازه‌ی مذکور است، زیرا همواره خطای گرد کردن کوچک‌تر از 10^{-n} خواهد بود.
- از سوی دیگر، محدودیت بازه‌ی اعداد قابل تولید با استفاده از تعداد محدودی بایت حافظه مهم‌ترین نقطه‌ی ضعف آن است.
- عموماً برای محاسباتی که به دفعات زیادی اعمال «جمع و تفریق» لازم است مناسب است، مانند حسابداری مالی بانک‌ها

انواع داده‌های درونی: اعداد اعشاری با ممیز شناور

اعداد اعشاری با ممیز شناور (floating-point float numbers):
در این روش همواره تعدادی ثابت از ارقام **با معنی** نگه داشته می‌شوند. به عبارت دیگر، از یک عدد صحیح با تعداد ارقام m برای ذخیره‌ی ارقام بامعنی و از یک عدد صحیح دیگر برای ذخیره‌ی مرتبه‌ی بزرگی استفاده می‌گردد:

$$f = d_۱ d_۲ d_۳ \dots d_m \times ۱۰^n$$

مهم‌ترین ویژگی‌های این روش عبارتند از:

- نقطه‌ی قوت این روش در گستردگی بازه‌ی اعداد قابل تولید با استفاده از تعداد محدودی بایت حافظه است.
- از سوی دیگر، انباشته شدن خطای گرد کردن می‌تواند به نتایج بی‌اعتبار بیانجامد.
- عموماً برای محاسباتی که به دفعات زیادی اعمال «ضرب و تقسیم» لازم است مناسب است، مانند محاسبات علمی و مهندسی

انواع داده‌های درونی: اعداد اعشاری با ممیز شناور

زبان‌های برنامه‌نویسی معمولاً چند نوع از اعداد اعشاری با ممیز شناور با بازه‌های متفاوت را پشتیبانی می‌کنند.

نام	تعداد بایت	تعداد بیت s+coeff.+exp.	تعداد ارقام بامعنی در مبنای ۱۰	بازه‌ی توان در مبنای ۱۰	C	Fortran	Python
half	2	$1+10+5$	3 or 4	± 5	———	———	`numpy.float16`
single	4	$1+23+8$	6 to 9	± 38	`float`	`REAL(KIND=4)`	`numpy.float32`
double	8	$1+52+11$	15 to 17	± 308	`double`	`REAL(KIND=8)`	`float` `numpy.float64`
quadruple	16	$1+112+15$	33 to 36	$\pm 4{,}932$	`long double`	`REAL(KIND=16)`	`numpy.float128`

- تعداد ارقام بامعنی اعداد اعشاری با دقت مضاعف (double precision) بین ۱۵ تا ۱۷ رقم است. این بدین معناست که اگر یک عدد اعشاری دارای ۱۵ رقم بامعنی در مبنای ۱۰ باشد، با تبدیل آن از مبنای ۱۰ به معادل دودویی دقت مضاعف و سپس تبدیل برعکس به مبنای ۱۰، عدد اولیه به دست می‌آید.
- در هنگام کار با اعداد اعشاری با ممیز شناور همواره باید مراقب خطای گرد کردن (round-off error) بود.

انواع داده‌های درونی: اعداد اعشاری با ممیز شناور

عملیات ریاضی و مقایسه‌ی اعداد اعشاری با ممیز شناور نیازمند توجه بیشتر است:

```
>>> a = "0.12345678901234567890123456789012345678901234567890"
>>> a
'0.12345678901234567890123456789012345678901234567890'
>>>
>>> print("%.50f" % (numpy.float16(a)))
0.12347412109375000000000000000000000000000000000000
>>>
>>> print("%.50f" % (numpy.float32(a)))
0.12345679104328155517578125000000000000000000000000
>>>
>>> print("%.50f" % (numpy.float64(a)))
0.12345678901234567736988623209981597028672695159912
>>>
>>> print("%.50f" % (numpy.float128(a)))   # same as float64
0.12345678901234567736988623209981597028672695159912
>>>
>>> print("%.50f" % (float(a)))   # same as float64
0.12345678901234567736988623209981597028672695159912
>>>
```

انواع داده‌های درونی: اعداد اعشاری با ممیز شناور

عملیات ریاضی و مقایسه‌ی اعداد اعشاری با ممیز شناور نیازمند توجه بیشتر است:

```
>>> x_int = 12345678901234567890
>>> x_int
12345678901234567890
>>>
>>> x_int + 1 > x_int
True
>>>
>>>
>>> x_float = 12345678901234567890.0
>>> x_float
1.2345678901234567e+19
>>>
>>> x_float + 1 > x_float
False
>>> x_float + 1000 > x_float
False
>>> x_float + 1000 == x_float
True
>>>
```

انواع داده‌های درونی: اعداد اعشاری با ممیز شناور

عملیات ریاضی و مقایسه‌ی اعداد اعشاری با ممیز شناور نیازمند توجه بیشتر است:

```
>>> x = numpy.float16(0.0)
>>> for i in range(0, 100):
...     x = x + numpy.float16(0.01)
...
>>> x
0.98828
>>> (1.0 - x) * 100
1.171875    # relative error in %
>>>
>>>
>>> # the same calculations using single-precision floats
>>> x = numpy.float32(0.0)
>>> for i in range(0, 100):
...     x = x + numpy.float32(0.01)
...
>>> x
0.99999934
>>> (1.0 - x) * 100
6.556510925292968e-05    # relative error in %
>>>
```

انواع داده‌های درونی: اعداد اعشاری با ممیز شناور

عملیات ریاضی و مقایسه‌ی اعداد اعشاری با ممیز شناور نیازمند توجه بیشتر است:

```
>>> # the same calculations using double-precision floats
>>> x = 0.0
>>> for i in range(0, 100):
...     x = x + 0.01
...
>>> x
1.0000000000000007
>>> (1.0 - x) * 100
-6.661338147750939e-14    # relative error in %
>>>
```

انواع داده‌های درونی: اعداد اعشاری با ممیز شناور

عملیات ریاضی و مقایسه‌ی اعداد اعشاری با ممیز شناور نیازمند توجه بیشتر است:

```
>>> x = 3.3
>>> y = 1.1 + 1.1 + 1.1
>>>
>>> x
3.3
>>> y
3.3000000000000003
>>>
>>> y == x
False
>>>
>>> abs(y - x) < 1.0e-9
True
>>>
```

انواع داده‌های درونی: اعداد اعشاری با ممیز شناور

عملیات ریاضی و مقایسه‌ی اعداد اعشاری با ممیز شناور نیازمند توجه بیشتر است:

```
>>> x = 1.1 * 1000000
>>> x
1100000.0
>>>
>>> y = 0.0
>>> for i in range(0, 1000000):
...     y = y + 1.1
...
>>> y
1099999.9999886872
>>>
>>> y == x
False
>>>
>>> abs(y - x) < 1.0e-9
False
>>>
>>> abs(y - x) / (abs(y) + abs(x) + 1.0e-15) < 1.0e-9
True
>>>
```

انواع داده‌های درونی: جمع‌بندی

۱. در پایتون ۳، دقت اعداد صحیح درونی (built-in) نامحدود است. در نتیجه، عملیات ریاضی بر روی اعداد صحیح و مقایسه‌ی آن‌ها بسیار سرراست است.

۲. در پایتون ۳، اعداد اعشاری درونی به صورت ممیز شناور با دقت مضاعف (double precision) ذخیره می‌شوند. در نتیجه:
   - دقت اعداد اعشاری درونی محدود است.
   - عملیات ریاضی بر روی اعداد اعشاری همواره باعث انباشته شدن خطای گردکردن می‌گردد.
   - مقایسه‌ی مقادیر دو عدد اعشاری در بیشتر مواقع به نتیجه‌ی نادرست منتهی می‌شود.
   - به جای در نظر گرفتن اختلاف مطلق مقادیر دو عدد اعشاری، اختلاف نسبی آن‌ها مهم است.

با آرزوی موفقیت ...