



# Software Development 1

## Strings

TUDublin –Tallaght Campus

E.Costelloe

# Strings

String - An ordered collection of characters used to store and represent text information.

**ASCII** (American Standard Code for Information Interchange) is the most common **format** for text files in computers and on the Internet. In an **ASCII** file, each alphabetic, numeric, or special character is represented with a 7-bit binary number (a string of seven 0s or 1s). 128 possible characters are defined.

Windows NT and 2000 uses a newer code, [Unicode](#), (supports more characters).

# ASCII Table - <http://www.asciitable.com/>

Dec	Hx	Oct	Char	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
0	0	000	<b>NUL</b> (null)	32	20	040	&#32;	<b>Space</b>	64	40	100	&#64;	<b>@</b>	96	60	140	&#96;	<b>`</b>
1	1	001	<b>SOH</b> (start of heading)	33	21	041	&#33;	<b>!</b>	65	41	101	&#65;	<b>A</b>	97	61	141	&#97;	<b>a</b>
2	2	002	<b>STX</b> (start of text)	34	22	042	&#34;	<b>"</b>	66	42	102	&#66;	<b>B</b>	98	62	142	&#98;	<b>b</b>
3	3	003	<b>ETX</b> (end of text)	35	23	043	&#35;	<b>#</b>	67	43	103	&#67;	<b>C</b>	99	63	143	&#99;	<b>c</b>
4	4	004	<b>EOT</b> (end of transmission)	36	24	044	&#36;	<b>\$</b>	68	44	104	&#68;	<b>D</b>	100	64	144	&#100;	<b>d</b>
5	5	005	<b>ENQ</b> (enquiry)	37	25	045	&#37;	<b>%</b>	69	45	105	&#69;	<b>E</b>	101	65	145	&#101;	<b>e</b>
6	6	006	<b>ACK</b> (acknowledge)	38	26	046	&#38;	<b>&amp;</b>	70	46	106	&#70;	<b>F</b>	102	66	146	&#102;	<b>f</b>
7	7	007	<b>BEL</b> (bell)	39	27	047	&#39;	<b>'</b>	71	47	107	&#71;	<b>G</b>	103	67	147	&#103;	<b>g</b>
8	8	010	<b>BS</b> (backspace)	40	28	050	&#40;	<b>(</b>	72	48	110	&#72;	<b>H</b>	104	68	150	&#104;	<b>h</b>
9	9	011	<b>TAB</b> (horizontal tab)	41	29	051	&#41;	<b>)</b>	73	49	111	&#73;	<b>I</b>	105	69	151	&#105;	<b>i</b>
10	A	012	<b>LF</b> (NL line feed, new line)	42	2A	052	&#42;	<b>*</b>	74	4A	112	&#74;	<b>J</b>	106	6A	152	&#106;	<b>j</b>
11	B	013	<b>VT</b> (vertical tab)	43	2B	053	&#43;	<b>+</b>	75	4B	113	&#75;	<b>K</b>	107	6B	153	&#107;	<b>k</b>
12	C	014	<b>FF</b> (NP form feed, new page)	44	2C	054	&#44;	<b>,</b>	76	4C	114	&#76;	<b>L</b>	108	6C	154	&#108;	<b>l</b>
13	D	015	<b>CR</b> (carriage return)	45	2D	055	&#45;	<b>-</b>	77	4D	115	&#77;	<b>M</b>	109	6D	155	&#109;	<b>m</b>
14	E	016	<b>SO</b> (shift out)	46	2E	056	&#46;	<b>.</b>	78	4E	116	&#78;	<b>N</b>	110	6E	156	&#110;	<b>n</b>
15	F	017	<b>SI</b> (shift in)	47	2F	057	&#47;	<b>/</b>	79	4F	117	&#79;	<b>O</b>	111	6F	157	&#111;	<b>o</b>
16	10	020	<b>DLE</b> (data link escape)	48	30	060	&#48;	<b>0</b>	80	50	120	&#80;	<b>P</b>	112	70	160	&#112;	<b>p</b>
17	11	021	<b>DC1</b> (device control 1)	49	31	061	&#49;	<b>1</b>	81	51	121	&#81;	<b>Q</b>	113	71	161	&#113;	<b>q</b>
18	12	022	<b>DC2</b> (device control 2)	50	32	062	&#50;	<b>2</b>	82	52	122	&#82;	<b>R</b>	114	72	162	&#114;	<b>r</b>
19	13	023	<b>DC3</b> (device control 3)	51	33	063	&#51;	<b>3</b>	83	53	123	&#83;	<b>S</b>	115	73	163	&#115;	<b>s</b>
20	14	024	<b>DC4</b> (device control 4)	52	34	064	&#52;	<b>4</b>	84	54	124	&#84;	<b>T</b>	116	74	164	&#116;	<b>t</b>
21	15	025	<b>NAK</b> (negative acknowledge)	53	35	065	&#53;	<b>5</b>	85	55	125	&#85;	<b>U</b>	117	75	165	&#117;	<b>u</b>
22	16	026	<b>SYN</b> (synchronous idle)	54	36	066	&#54;	<b>6</b>	86	56	126	&#86;	<b>V</b>	118	76	166	&#118;	<b>v</b>
23	17	027	<b>ETB</b> (end of trans. block)	55	37	067	&#55;	<b>7</b>	87	57	127	&#87;	<b>W</b>	119	77	167	&#119;	<b>w</b>
24	18	030	<b>CAN</b> (cancel)	56	38	070	&#56;	<b>8</b>	88	58	130	&#88;	<b>X</b>	120	78	170	&#120;	<b>x</b>
25	19	031	<b>EM</b> (end of medium)	57	39	071	&#57;	<b>9</b>	89	59	131	&#89;	<b>Y</b>	121	79	171	&#121;	<b>y</b>
26	1A	032	<b>SUB</b> (substitute)	58	3A	072	&#58;	<b>:</b>	90	5A	132	&#90;	<b>Z</b>	122	7A	172	&#122;	<b>z</b>
27	1B	033	<b>ESC</b> (escape)	59	3B	073	&#59;	<b>;</b>	91	5B	133	&#91;	<b>[</b>	123	7B	173	&#123;	<b>{</b>
28	1C	034	<b>FS</b> (file separator)	60	3C	074	&#60;	<b>&lt;</b>	92	5C	134	&#92;	<b>\</b>	124	7C	174	&#124;	<b> </b>
29	1D	035	<b>GS</b> (group separator)	61	3D	075	&#61;	<b>=</b>	93	5D	135	&#93;	<b>]</b>	125	7D	175	&#125;	<b>}</b>
30	1E	036	<b>RS</b> (record separator)	62	3E	076	&#62;	<b>&gt;</b>	94	5E	136	&#94;	<b>^</b>	126	7E	176	&#126;	<b>~</b>
31	1F	037	<b>US</b> (unit separator)	63	3F	077	&#63;	<b>?</b>	95	5F	137	&#95;	<b>_</b>	127	7F	177	&#127;	<b>DEL</b>

# ASCII & Unicode

Unicode- contains many additional characters, supports non-english letters etc.

ASCII is a simple form of Unicode text

The fundamental **str** string type, which handles ASCII text, works the same regardless of which version of Python you use and **str** also handles Unicode in Python 3.X, everything here will apply directly to Unicode processing too.

# String Basics

In Python, strings come with a powerful set of processing tools. Also Python has no distinct type for individual characters; instead, you just use one-character strings.

Python strings are categorized as *immutable sequences*, meaning that the characters they contain have a left-to-right positional order and that they cannot be changed in place.

```
my_string = "hello world"  
print(my_string)
```

# Single- and Double-Quoted Strings Are the Same

Around Python strings, single- and double-quote characters are interchangeable. That is, string literals can be written enclosed in either two single or two double quotes—the two forms work the same and return the same type of object. For example, the following two strings are identical, once coded:

```
>>> 'shrubbery', "shrubbery"  
('shrubbery', 'shrubbery')
```

The reason for supporting both is that it allows you to embed a quote character of the other variety inside a string without escaping it with a backslash. You may embed a single-quote character in a string enclosed in double-quote characters, and vice versa:

```
>>> 'knight"s', "knight's"  
('knight"s', "knight's")
```

Ref:Lutz M.

# Variables: Strings

- Strings can be added together (concatenation) with the + operator

```
string1 = "hello"  
string2 = "world"  
answer = string1 + string2  
print(answer)
```

**Output:**

**helloworld**

# Variables: Strings

- Strings can be added together (concatenation)

```
string1 = "hello"  
string2 = "world"  
answer = string1 + " " + string2  
print(answer)
```

---

**Output:**

hello world



# Variables: Strings

- Strings can be repeated, i.e. Multiplied, with the \* operator:

```
string1 = "hello"  
answer = string1 * 3  
print(answer)
```

**Output:**

**hellohellohello**

- Write the code to print hello 3 times but place a space between the hello strings, ensure there is no trailing space, i.e. at the end of the last hello string

# Use of Repetition

For example, to print a line of 80 dashes, you can count up to 80, or let Python count for you:

```
>>> print('----- ...more... ---') # 80 dashes, the hard way
```

```
>>> print('-' * 80) # 80 dashes, the easy way
```

# len function

- To find out the length of a string (len(stringname) returns a number, i.e. the number of characters, of type int)

```
string1 = "hello"  
length = len(string1)  
print(length)
```

**Output:**  
5

# String Methods <https://docs.python.org/3/library/string.html>

<https://docs.python.org/3/library/stdtypes.html#string-methods>

Strings provide a set of ***methods*** that implement more sophisticated text-processing tasks.

Methods are simply functions that are associated with and act upon particular objects.

The method call expression(syntax):

*object.method(arguments)*

in plain words, the method call expression means this:

Call *method* to process *object(variablename)* with *arguments*.

# String Methods

- To convert a string to Upper Case: (The reverse is true of **lower()** method)

```
string1 = "hello"  
string1 = string1.upper()  
print(string1)
```

Output:  
HELLO

- If the method computes a result, it will also come back as the result of the entire method call expression. For example:
- The result of the expression string1.upper() is “HELLO” which is then placed in **string1**
- Write the line of code that would retain the original value of string1 and store the upper case version in a variable called string2

# String Methods


```
string1 = "hello"  
output = string1.capitalize()  
print(output)
```

**Output:**  
**Hello**

capitalize() returns a copy of the string with its first character capitalized and the rest lowercase


# String Formatting

- We previously mentioned this method earlier:
- Concatenation VS Formatting
- With concatenation you may create a new string from existing strings and other data types, but be careful you must convert other numeric types to strings first before adding /concatenating them



```
name = "John"  
age = 21  
output = "hello " + name + " you are " + str(age) + " years old"  
print(output)
```

Formatting with format method:



```
name = "John"  
age = 21  
output = "hello {0} you are {1} years old".format(name, age)  
print(output)
```

# String Formatting

```
name = "John"  
age = 21  
output = "hello " + name + " you are " + str(age) + " years old"  
print(output)
```

This creates 5 strings, concatenates them and stores them in output.

*-could reduce performance on large concatenations*

*- Numeric and Boolean values have to be cast to strings*

```
name = "John"  
age = 21  
output = "hello {0} you are {1} years old".format(name, age)  
print(output)
```

Performance gains and readability



# String Methods

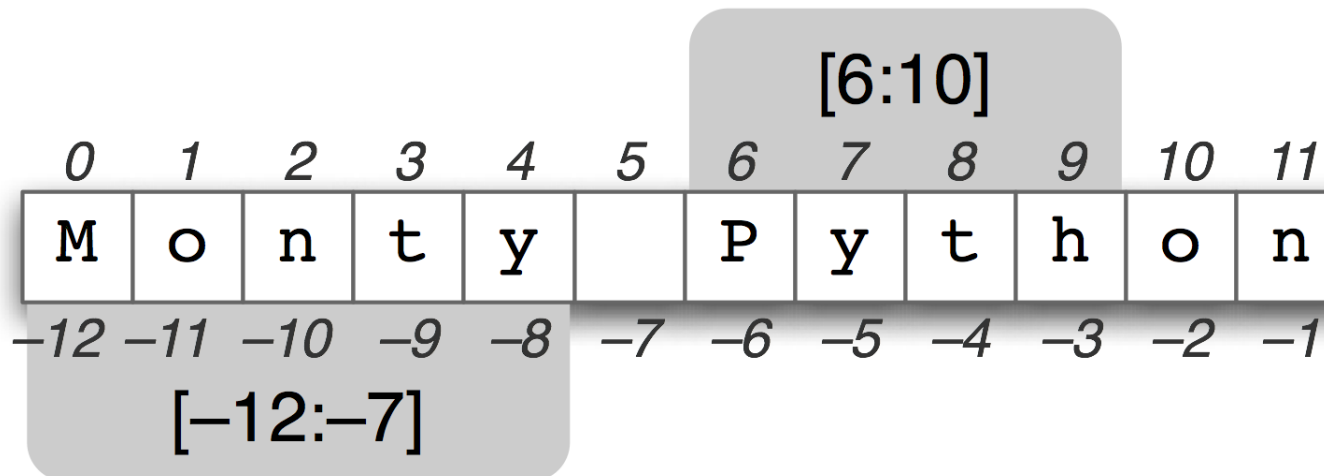
- There are more methods, for all types that we use:

Visit : <https://docs.python.org/3/library/stdtypes.html>

# Strings in detail

Strings in Python do not consist of characters (as Python does not support characters)

- A string can be considered as a sequence of strings each with a length of 1
- These individual strings of length 1 can be accessed by using the name of the string followed by[] with the position of the character(index- integer) to be accessed placed in the [].
- An **index**, in a string, refers to a position within the string. **Python** strings can be thought of as lists of characters; each character is given an **index** from zero (at the beginning) to the length minus one (at the end)
- First lets look a String Indexing in Python (and most other languages)



# String in detail

0	1	2	3	4	5	6	7	8	9	10	11
M	o	n	t	y		P	y	t	h	o	n
-12	-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1

[6:10]

[-12:-7]

Consider the following:

```
my_string = "Monty Python"  
print(my_string[0])
```

Output:  
M

Name of string followed by [] with index of 0 specified in the []

# String in detail

0	1	2	3	4	5	6	7	8	9	10	11
M	o	n	t	y		P	y	t	h	o	n
-12	-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1

[6:10]

[-12:-7]

Consider the following:

In Python, we can also index backwards, from the end, negative indexes count back from the right and start at -1

```
my_string = "Monty Python"  
print(my_string[-1])
```



Output:  
n

# String in detail

0	1	2	3	4	5	6	7	8	9	10	11
M	o	n	t	y		P	y	t	h	o	n
-12	-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1

[6:10]

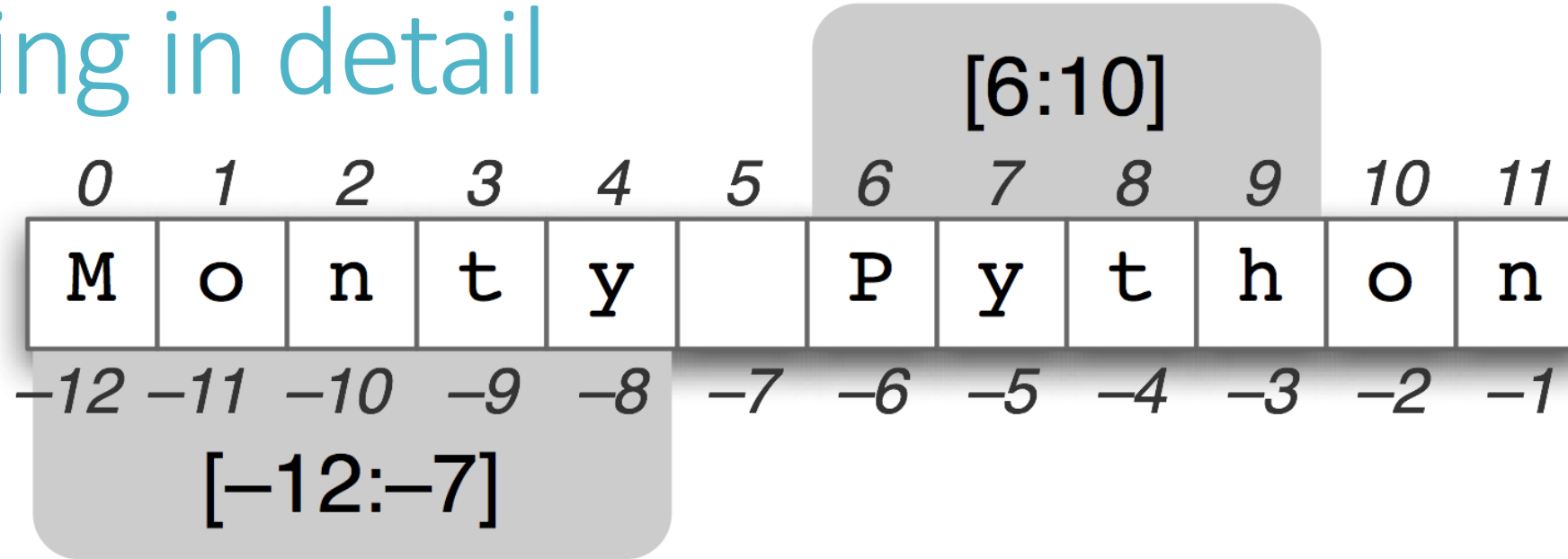
[-12:-7]

```
my_string = "Monty Python"  
print(my_string[-1])
```

```
my_string = "Monty Python"  
print(my_string[len(my_string) - 1])
```

Output: (Same results)  
n

# String in detail



The relationship between the last string item and length is:

```
my_string = "Monty Python"
print(my_string[len(my_string) - 1])
```

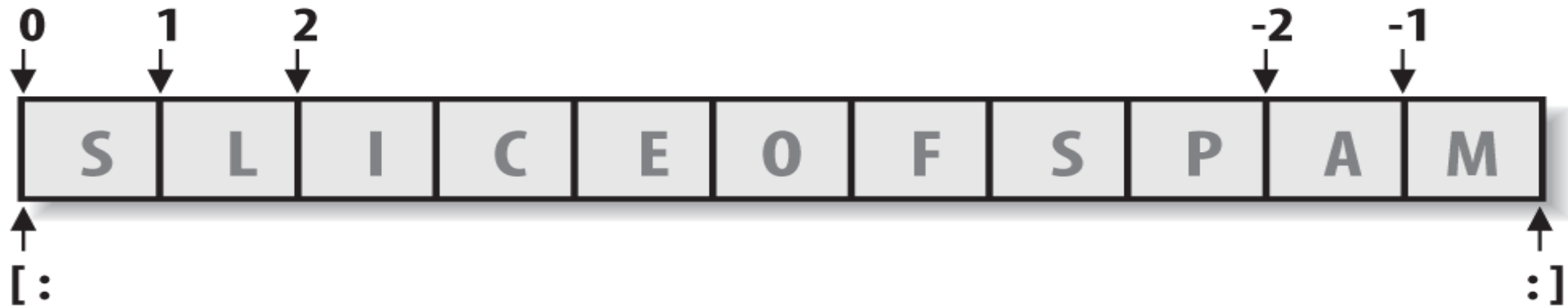
Output:  
n

Note that we can use an arbitrary expression in the square brackets, not just a hard coded number literal, anywhere that Python expects a value we can use a literal, a variable, or any expression we wish, it must evaluate to an integer in this instance as is used as an index.

# Indexing and Slicing

`[start:end]`

*Indexes refer to places the knife “cuts.”*

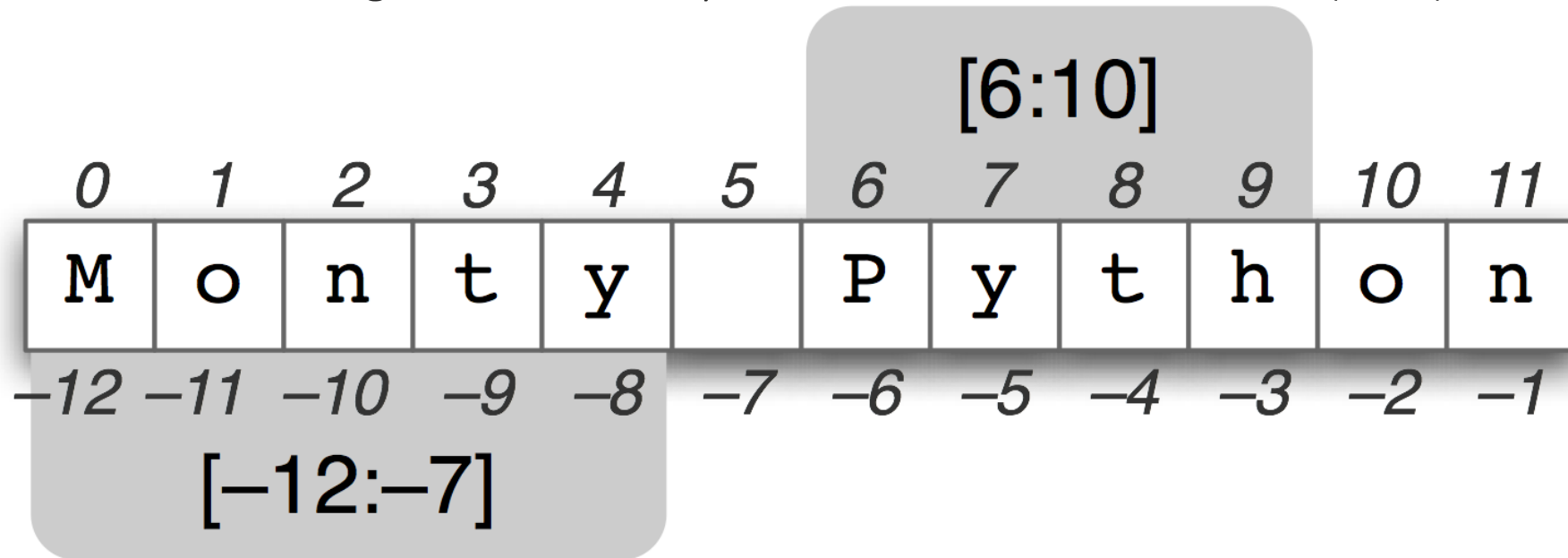


*Defaults are beginning of sequence and end of sequence.*

Slicing returns a new object containing the section identified by the offset pair. The Left offset is the lower boundary(inclusive) and the right offset is the upper boundary(non-inclusive)  
If omitted , the left and right boundaries default to 0 and the length of the object you are slicing, respectively.

# Strings- Slicing

Slicing – a form of indexing which is a way to extract an entire section(slice) of a string in a single step



Consider the following:

```
my_string = "Monty Python"  
print(my_string[6:10])
```

Output:  
Pyth



Slice of my\_string from position(index) 6 through to but not including 10



# Slicing a string

Syntax: `stringname[x:y]` returns the part of the string from the x-th character to the y-th character including the first(x) but excluding the last(y)

If you omit the first index – the slice starts from the beginning of the string, index 0

If you omit the last index – the slice goes to the end of the string

What is the output of the following?

```
print(my_string[:5])  
print(my_string[6:])  
print(my_string[:])
```

# Immutability

Every string operation is defined to produce a new string as its result, because strings are *immutable in Python*—*they cannot be changed* in place after they are created. In other words, you can never overwrite the values of immutable objects. For example, you can't change a string by assigning to one of its positions, but you can always build a new one and assign it to the same name. Because Python cleans up old objects as you go (garbage collection) this isn't as inefficient as it may sound:

```
my_name = "joe Bloggs"  
my_name[0] = "J"
```

*# Immutable objects cannot be changed*

**ERROR GENERATED:**

Traceback (most recent call last):

File "C:/Users/Eileen/PycharmProjects/week2variablesnotes/variablesnotes.py", line 225, in <module>

```
my_name[0] = "J"
```

TypeError: 'str' object does not support item assignment

Every object in Python is classified as either immutable (unchangeable) or not.

In terms of the core types, *numbers and strings are immutable*

*Ref(Lutz,M.)*

# How to edit a string -String method- replace()

The string **replace** method performs global searches and replacements; it acts on the subject that it is attached to and called from:

```
my_name = "joe Bloggs"  
my_name = my_name.replace("j", "J")    # Replace occurrences of a string in my_name with another  
print("My name is:", my_name)
```

## Output:

My name is: Joe Bloggs

Again, despite the name of the string method, we are not changing the original string here, but creating a new string as the result and reassigning it to my\_name, —because strings are immutable, this is the only way this can work.

# String finding a substring using **in** operator

- First, examine if the substring (in this case a space " "), is in the string

Membership operators are operators used to validate the membership of a value. It test for membership in a sequence, such as strings etc..

**in operator** : The 'in' operator is used to check if a value exists in a sequence or not. Evaluates to true if it finds a **value** in the specified sequence and false otherwise.

```
my_name = "Eileen Costelloe"  
print(" " in my_name)
```

Output:

True

# Boolean

# String finding a substring

- First, examine if the substring (in this case a space " "), is in the string

```
my_name = "EileenCostelloe"  
print(" " in my_name)
```

Output:

False      # Boolean

```
my_name = "eileencostelloe"  
print("co" in my_name)
```

Output:

True      # Boolean

# String finding a substring

- First, examine if the substring (in this case a space “Co”), is in the string

```
my_name = "eileencostelloe"  
print("Co" in my_name)
```

Output:

```
False      # Boolean
```

# String a location of a substring- index()

- Examine the string to find the location of a substring (in this case “ ”)

```
my_name = "Eileen Costelloe"  
space_loc = my_name.index(" ")  
print("Location of space:", space_loc)
```

**Output:**

**Location of space: 6**

- *index()* is an inbuilt function in Python, which searches for given element from start of the sequence and returns the lowest index where the element appears.
- This will throw a “ValueError” error if the substring is not found in the original string

# String a location of a substring

- Try this.... No space in string?

```
my_name = "EileenCostelloe"  
space_loc = my_name.index(" ")  
print("Location of space:", space_loc)
```

**Output:**  
?????



# String a location of a substring

- What would you expect this to output?

```
my_name = "Eileen Costelloe"  
wordLoc = my_name.index("Costelloe")  
print("Location of word:", wordLoc)
```

**Output:**

**Location of word: 7**

# Class work:

- Assuming that a person only has a first name and surname, separated by a single space, write a python script to
  - Take in from the user their full name.
  - Separate their name into two variables, f\_name and s\_name
  - Print the f\_name and s\_name individually.

# Class work:

```
# Input
persons_f_name = input("Please enter your full name, separated by a space:")

# Processing
loc_space = persons_f_name.index(" ")

f_name = persons_f_name[:loc_space]
s_name = persons_f_name[(loc_space + 1):]

f_name = f_name.capitalize()
s_name = s_name.capitalize()

# Output
print("First Name:", f_name)
print("Surname   :", s_name)
```