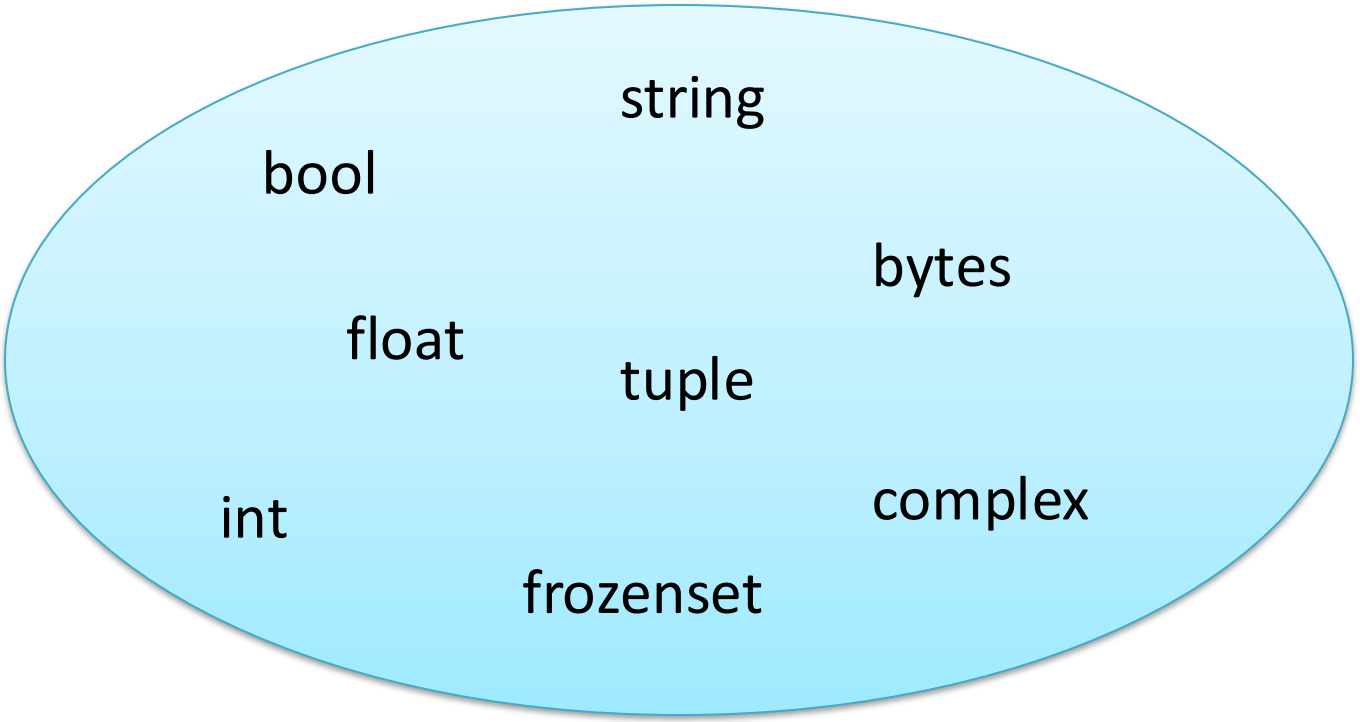
COSC 2306
Data Programming

Python Basics

Immutable types

Changes/reassign creates a new object



Try this:

```
text = "Python"
text2 = text
print(id(text))
print(id(text2))
print(text is text2)
print()

text += " is awesome"
print(id(text))
print(id(text2))
print(text is text2)
print()

print(text)
print(text2)
```

```
2
2
True

4
2
False

Python is awesome
Python
```

Mutable vs. immutable in functions

```
def updateList(list1):
    list1 += [10]

n = [5, 6]
print(id(n))
updateList(n)
print(n)
print(id(n))

>2
>[5, 6, 10]
>2
```

In-place modification, different from list1 = list1 + [10], which creates a new object

```
def updateNumber(n):
    print(id(n))
    n += 10

b = 5
print(id(b))
updateNumber(b)
print(b)

>2
>2  # pass-by-object-reference
>5  # immutable, from b = 5
    and not from inside the def
```

https://medium.com/@meghamohan/mutable-and-immutable-side-of-python-c2145cf72747

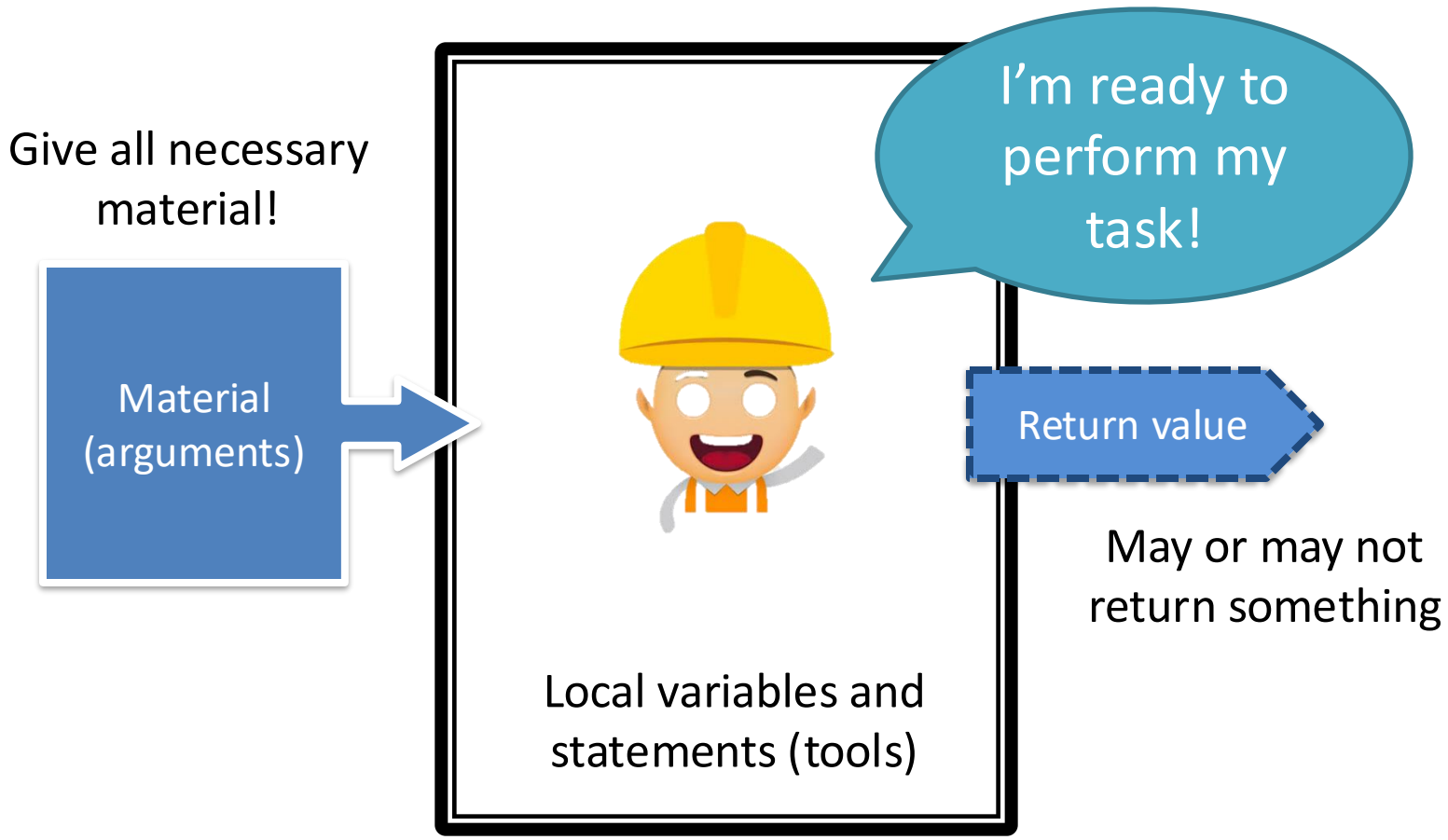
Mutable vs. immutable in functions

- ```
  def updateNumber(n):
      print(id(n))
      n += 10
      return n

  b = 5
  print(id(b))
  b = updateNumber(b)
  print(b)
  print(id(b))
  ```

```
>2
>2  #pass-by-object-reference
>15 #return
>3  #immutable, new object
```
```

Helpful functions



The `main` function

- Special function that is automatically invoked by the operating system when the program is executed
- Although Python does not require one (unlike C++ or Java), it is good practice to have one

```
def greeting(name):
    print("Hello ", name, "!", sep="")

def main():
    n = input("What's your name? ")
    greeting(n)

main()
```

Will this work?

```
def badsquare(x):
    y = x ** power
    return y

def main():
    power = 2
    result = badsquare(10)
    print(result)

main()
```

NameError: name “power” is not defined on line 2

Try this

```
def badsquare(x, power):
    y = x ** power
    return y

def main():
    power = 2
    result = badsquare(10, power)
    print(result)

main()
```

```
100
```

Functions with default values

In Python, functions can have default arguments

```
def isDivisible(x, y = 2):
    if x % y == 0:
        return True
    else:
        return False

print(isDivisible(6))
print(isDivisible(6,4))
```

```
>True
>False
```

Python Built-in Data Types

- Atomic data types:
  - int, float (arithmetic ops)
  - bool (true, false)
- Ordered collection data types (indexed):
  - list ([], mutable, heterogenous)
  - string ("", immutable, characters)
  - tuple ((), immutable, heterogenous)
- Unordered collection data types (hashing):
  - set ({}, mutable, heterogenous, immutable unique obj)
  - dictionary ({key:value}, mutable, heterogenous, unique key pairs, key typically immutable)

Lists

- A **list** is a sequential collection of Python data values
  - They can hold any type (and mix them-heterogenous)
  - Similar to strings, but **string** can only hold characters

```
vocabulary = ["iteration", "selection", "control"]
mixedlist = ["hello", 2.0, 5*2, [10, 20]]

print(vocabulary)
print(mixedlist)

print(len(vocabulary))
print(len(mixedlist))
```

>['iteration', 'selection', 'control']
>['hello', 2.0, 10, [10, 20]]
>3
>4

Accessing lists

- Lists can be accessed by indexing

```
numbers = [17, 123, 87, 34, 66, 8398, 44]
print(numbers[2])
print(numbers[-2])
print(numbers[len(numbers) - 1])
```

```
>87
>8398
>44
```

```
print(numbers[len(numbers)])
```

IndexError: list index out of range

- Once an element is accessed, one can use it as a regular variable of that type

```
alist = [3, 67, "cat", [56, 57, "dog"], [ ], 3.14, False]
print(alist[2][0])
print(alist[2].upper())
print(alist[2])
```

```
>c
>CAT
>cat
```

Other list operations

Name	Purpose	Example

Membership (in)	Test if an item is in a list	`fruit = ["apple", "orange", "banana", "cherry"]`
		`print("apple" in fruit)`
		`>True`

Concatenation (+)	Merge two lists	`print([1, 2] + [3, 4])` `>[1, 2, 3, 4]`
Repetition (*)	Repeat a list	`print([0] * 4)` `>[0, 0, 0, 0]`

Slicing ([:])	Access a range of items	`alist = ['a', 'b', 'c', 'd']`
		`print(alist[1:3])`
		`print(alist[:4])`
		`>['b', 'c']`
		`>['a', 'b', 'c', 'd']`

Deletion (del)	Remove element(s)	`alist = ['a', 'b', 'c', 'd']`
		`del alist[1] #['a', 'c', 'd']`
		`del alist[:1] #['b', 'c', 'd']`

list operation example

```
alist = [1,2]
print(id(alist))

blist = [3,4]
print(id(blist))

clist = (alist + blist)
print(clist)
print(id(clist))
```

```
2
3
[1,2,3,4]
4
```

Lists are mutable

```
fruit = ["banana", "apple", "cherry"]
print(fruit)

fruit[0] = "pear"
fruit[-1] = "orange"
print(fruit)
```

>['banana', 'apple', 'cherry']
>['pear', 'apple', 'orange']

You can use id(fruit) to verify that the identifier of fruit stayed the same

```
astring = "Hello"
astring[0] = "W"
print(astring)
```

TypeError: 'str' object does not support item assignment

Aliasing vs. cloning

```
a = [81, 82, 83]
b = a
print(a is b)
print(a == b)
```

>True
>True

```
a[1] = 0
print(a)
print(b)
```

>[81, 0, 83]
>[81, 0, 83]

```
a = [81, 82, 83]
b = a[:]
print(a is b)
print(a == b)
```

>False
>True

```
a[1] = 0
print(a)
print(b)
```

>[81, 0, 83]
>[81, 82, 83]

More list methods

Method	Parameters	Result	Description
append	item	mutator	Adds a new item to the end of a list
insert	position, item	mutator	Inserts a new item at the position given
pop	none	hybrid	Removes and returns the last item
pop	position	hybrid	Removes and returns the item at position
sort	none	mutator	Modifies a list to be sorted
reverse	none	mutator	Modifies a list to be in reverse order
index	item	return idx	Returns the position of first occurrence of item
count	item	return ct	Returns the number of occurrences of item
remove	item	mutator	Removes the first occurrence of item

```
mylist = mylist.sort()?
```